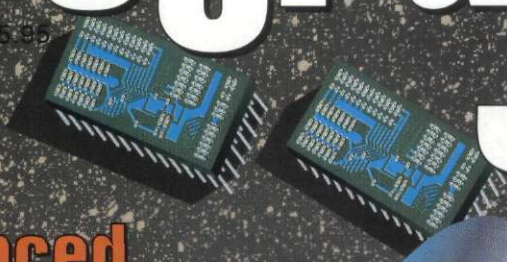
HARDWARE FOR DEVELOPERS

May/June 1991 Volume 2, 3

Programmer's Journal

95 Canada $5.95



Enhanced Debugging with Hardware

page 26

SCSI Portability with OOPs

page 32

Handling Numeric Exceptions

page 40

Pixel Panning

page 21

More Code for Less Money

page 14

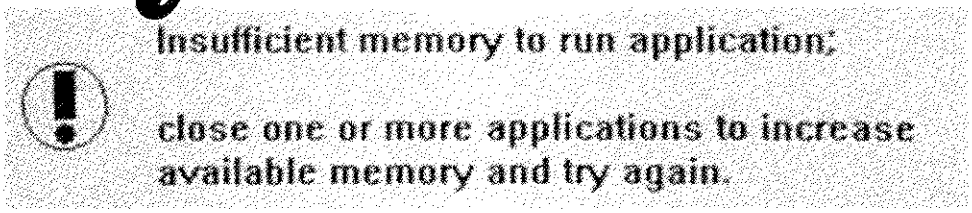
Hashing Functions

page 69

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

386 and 486 Windows users:

DOS Lives. Despite what you may have read.



Windows 3.0 may have been a big step forward for some programs.

But it was a big step backward for DOS. Suddenly, it was 1987 all over again. Not enough room for DOS programs to run because TSRs, utilities, drivers and buffers were taking up room your DOS programs need.

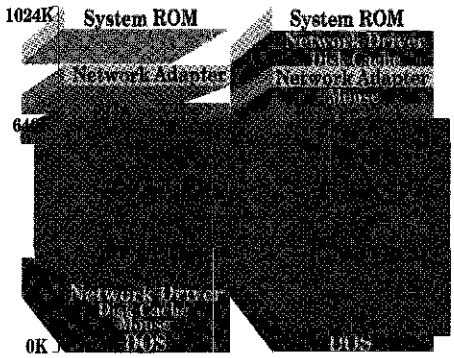
QEMM 5.1 to the rescue

Now, we've updated QEMM to provide additional memory for DOS programs within Windows 3.0.

QEMM 5.1 works with the built-in capabilities of your 80386 or i486 processor to find and recover unused memory segments. As you can see in the chart, there are gaps in your PC's memory usage above 640K. QEMM 5.1 fills those gaps and provides more room for your DOS programs to run.

And of course, QEMM 5.1 still works with DOS when you're not running Windows. You get all the same benefits: up to 130K more memory to run the new generation of memory-hungry programs; space for larger spreadsheets and database files.

When Windows 3.0 says there's not enough room to run your DOS programs, it's just trying to tell you it needs QEMM 5.1.



Before *After*

Unused memory is like hidden gold in your PC. QEMM finds it and makes it available.

Introducing DESQview 2.3 and DESQview 386 2.3 for users of Windows 3.0

They said it couldn't be done, but DESQview 2.3 can run Windows 3.0 programs. Not just in Windows "Real mode" but in "Standard" mode. That means programs can be up to 16MB.

And it can run DOS programs and DOS-extended programs i.e., 1-2-3 Release 3, side-by-side.

DESQview 386 2.3 does all that and more. It lets you run 386 DOS extended programs like AutoCAD 386 and IBM Interleaf side-by-side.

DOS, extended DOS; Windows—whatever standards you set, we will support. We're committed to helping you get the most out of your hardware and software today. And tomorrow.

QEMM-386 System Requirements: 80386-based PCs and PS/2s and PCs with 80386 add-in boards. Operating system: PC DOS 2.0-4.0, MS DOS 2.0-3.3, Windows 3.0. Conventional memory requirement 1.5K.
DESQview System Requirements: IBM Personal Computer and 100% compatibles (with 8086, 8088, 80286, or 80386 processors) with monochrome or color display; IBM Personal System/2 • Memory: 640K recommended; for DESQview itself 0-145K • Expanded Memory (Optional): expanded memory boards compatible with the Intel AboveBoard; enhanced expanded memory boards compatible with the AST RAMpage; EMS 4.0 expanded memory boards • Disk: two diskette drives or one diskette drive and a hard disk • Graphics Card (Optional): Hercules, IBM Color/Graphics (CGA). IBM Enhanced Graphics (EGA), IBM PS/2 Advanced Graphics (VGA) • Mouse (Optional): Mouse Systems, Microsoft and compatibles • Modem for Auto-Dialer (Optional): Hayes or compatible • Operating System: PC-DOS 2.0-4.0; MS-DOS 2.0-3.3 • Software: Most PC-DOS and MS-DOS programs; programs specific to Microsoft Windows 1.03-2.1, GEM 1.1-3.0, IBM TopView 1.1 • Media: DESQview is available on either 5-1/4" or 3-1/2" floppy diskette.

Trademarks: Windows, MS-DOS: Microsoft Corporation; PS/2, Interleaf, TopView: IBM Corporation; 80386, i486, AboveBoard: Intel Corporation; 1-2-3: Lotus Development Corporation; AutoCAD 386: Autodesk, Inc; RAMpage: AST Research; Hercules; Mouse Systems; Hayes; GEM: Digital Research, Inc.

©1990 Quarterdeck Office Systems

Yes! I need increased DOS productivity in Windows!

Qty	Product	5-1/4	3-1/2	Each	Totals
	QEMM 386 5.1			$99.95	
	DESQview 386 2.3 (includes QEMM)			$219.95	
	DESQview 2.3			$129.95	
				Shipping & Handling $5 in USA/$10 outside USA	
				California Residents add 6.75%	
				Grand Total	

Payment method ☐ Visa ☐ MasterCard
Expiration ____/____
Card # _____
Name _____
Address _____
City _____
State ____ Zip _____

Please allow 3 weeks for delivery

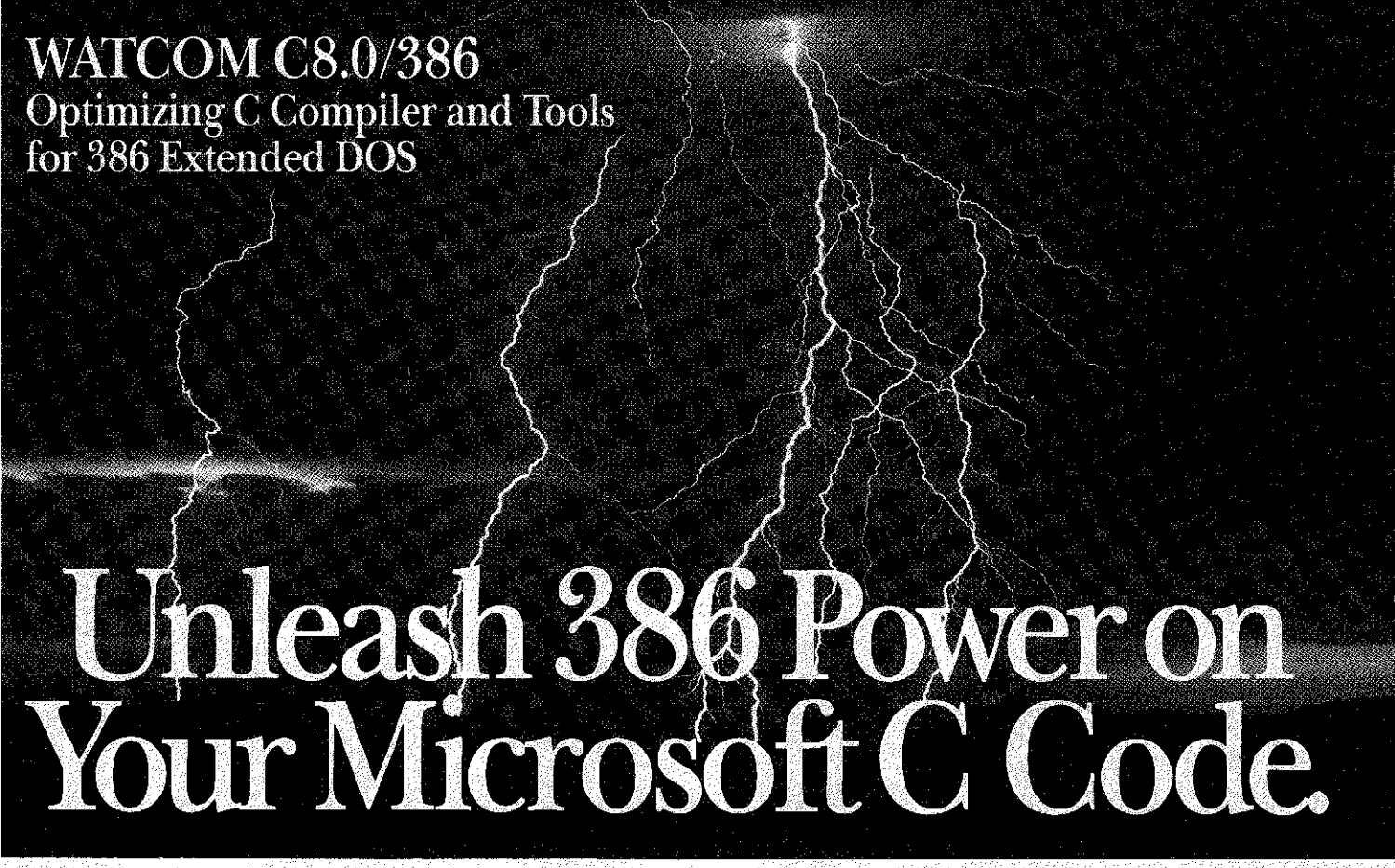
Quarterdeck

Quarterdeck Office Systems, 150 Pico Blvd., Santa Monica, CA 90405 (213) 392-9851 Fax: (213) 399-3802

Circle 1 on Reader Service Card

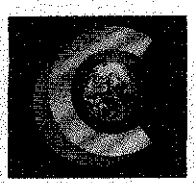
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

WATCOM C8.0/386
Optimizing C Compiler and Tools for 386 Extended DOS



Unleash 386 Power on Your Microsoft C Code.

- Interactive source-level debugger
- Generates high-performance code for 32-bit protected mode
- Microsoft source and library compatible
- Fast, tight code
- Profiler
- Protected-mode version of compiler
- Graphics library
- 100% ANSI C and SAA compatible
- Run-time compatible with WATCOM FORTRAN 77/386



Experts Agree on WATCOM C:

"When Novell went looking for a 32-bit compiler for use with the NetWare 386 developer's kit, the company selected WATCOM's...It's clear that Novell chose wisely; this product is a winner."
Fred Hommel, BYTE, December 1989

"WATCOM C/386 is a fantastic new ANSI C compatible compiler for 386-based PC's...If you have written your application in Microsoft C, you will love this compiler."
J. Richard Hines, Electronic Test, December 1989

"Microsoft library- and source- compatibility makes WATCOM C7.0/386 ideal for porting DOS applications to 32-bit native mode. This compiler enables full 386 performance without 640K limitations."
Richard M. Smith, President, Phar Lap Software, Inc.

"WATCOM is definitely the leader in object-level optimizations...For flat-out executable speed,...WATCOM C showed shining performance."
Computer Language, February 1989

WATCOM C8.0/386 Professional

• 100% ANSI C optimizing compiler • Protected-mode version of compiler • 386 run-time library object code • Windowed source level debugger • Profiler • Editor • 386 graphics library • MAKE • Linker • Object-code librarian • Object-code disassembler • Supports Phar Lap and ERGO DOS extenders

WATCOM F77/386

• 32-bit optimizing FORTRAN compiler based on WATCOM C technology • Full ANSI FORTRAN 77 plus extensions • Includes WATCOM C development tools: Windowed source-level debugger, Profiler, Editor, MAKE, Linker, Object-code librarian and Object-code disassembler • Protected-mode version of compiler • Run-time compatible with WATCOM C8.0/386 • Supports Phar Lap and ERGO DOS extenders

WATCOM **1-800-265-4555**

415 Phillip Street, Waterloo, Ontario, Canada N2L 3X2 Tel. (519) 886-3700 Fax (519) 747-4971

WATCOM C is a trademark of WATCOM Systems Inc. Trademarked names are the properties of their respective owners.
© Copyright 1990 WATCOM Products Inc.

Contact this advertiser directly.

PRINTED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Don't leave your customers HANGING. . .

BEFORE YOU SHIP ANY PROGRAM, CHECK IT WITH

COMPUTER LANGUAGE PRODUCTIVITY AWARD 1990

BOUNDS-CHECKER™

WINNER PC MAGAZINE BEST OF 1990

FINDS OUT-OF-BOUNDS MEMORY ACCESSES *AUTOMATICALLY*

Your program may have 10,000 to a million lines of code. It may occasionally hang mysteriously or it may appear to run flawlessly every time. But under DOS, how can you ever be sure that your program is not corrupting memory it does not own? The only way to be 100% sure is to BOUNDS-CHECK before you ship.

To use BOUNDS-CHECKER you build your program with debugging information (we support most compilers including Microsoft, Borland & JPI). Then you just type <BC file-name>. BOUNDS-CHECKER sets up the 386™/i486 for protection and lets your program fly. If your program accesses memory it does not own or overwrites its own code, BOUNDS-CHECKER pops up displaying the offending SOURCE-LINE or instruction.

Programming under DOS is a gamble, so why not stack the odds in your favor--CALL TODAY.

(603) 888-2386

OTHER FINE NU-MEGA PRODUCTS. . .

The ultimate systems debugger.

Debug:
- Interrupt routines
- Device drivers
- T&SRs
- ROMS
- Applications
- Overlays

Soft-ICE™

Features:
- Break out of a hung program
- Real time Break-Points
- Back-Trace history
- Works with other debuggers

If you are debugging an application, Soft-ICE is seamlessly integrated with BOUNDS-CHECKER so you can easily go back and forth between BOUNDS-CHECKing and debugging: a combination many programming professionals can't live without.

1988 BYTE AWARD OF DISTINCTION

Run CODEVIEW for Windows on a single monitor
CV/1.....$129
• Runs in a window • No annoying flash •
• Runs on any display that supports windows •

Call by 4:00 PM EST TODAY and ask us to EXPRESS you an info packet. In most cases you will receive it by 10:30 AM tomorrow. (USA only)

All Nu-Mega products require a 386, 386SX or 486. MS-DOS and Codeview are trademarks of Microsoft Corp. 386 is a registered trademark of Intel Corp. Nu-Mega, BOUNDS-CHECKER, Soft-ICE and CV/1 are trademarks of Nu-Mega Technologies, Inc.

Nu-Mega™ TECHNOLOGIES INC

P.O. Box 7780 • Nashua, NH • 03060-7780 U.S.A.
(603) 888-2386 • Fax (603) 888-2465

BOUNDS-CHECKER $249
Soft-ICE $386
Special Offer
Buy BC & S-ICE .....Save $100

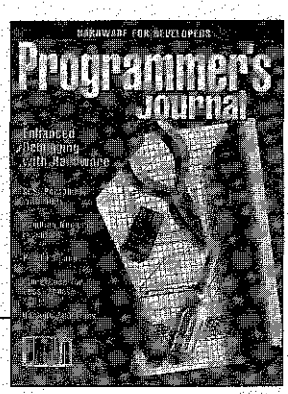
30 Day Money Back Guarantee

Circle 2 on Reader Service Card.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Programmer's Journal™

May/June 1991 Volume 9.3



Although the pioneers carved out their dreams with plows and axes, we construct our electronic world with silicon crystals, solder, and pliers. Whether you work with head or hands, this issue provides a sampling of the right tools for the task.

Cover Artist: Rob Schuster

FEATURES

GRAPHICS **21** **PIXEL PANNING** *By Peder Jungck*

Ever wondered how arcade games simulate motion and generate such smooth action images? Wonder no more. Peder explains how similar feats are possible on the EGA and VGA using fast panning to window onto a "larger than life" screen.

HARDWARE **26** **HARDWARE-ASSISTED DEBUGGERS** *By Brett Salter*

Have you ever hit a bug that stopped you cold? Were you unsure whether the problem was hardware or software? Don't be dismayed. Brett provides some help to the weary and the wary, describing hardware tools that can nail even the wiliest bugs.

HARDWARE **32** **OBJECT-ORIENTED SCSI PROGRAMMING** *By Brett Glass*

Although other PC hardware interfaces have long been standardized, SCSI programming has remained a free-for-all. Brett looks at the latest proposed Common Access Method (CAM) standard for SCSI and writes some portable code to interface with SCSI devices using Turbo Pascal objects.

C++ **40** **NUMERIC EXCEPTION HANDLING** *By Mark J. Barrenechea*

Floating-point errors have puzzled programmers for too long. Mark demystifies the IEEE 754 and 854 floating-point standards and explains how the Intel 80287/80387 and i860 chips fit into the puzzle. Learn some slick techniques for handling numeric exceptions in C++ with ease and elegance.

C **51** **VIRTUAL REALITIES** *By Stephen M. Kuhn*

When your text-based application can no longer deal with the memory-hungry real world, it's time to go virtual. Stephen shows you how to select and develop an application-supported virtual memory system that can take advantage of EMS paging and disk storage.

BASIC **60** **BASIC LIBRARIES** *By Murray L. Lesser*

Our BASIC ace shows how to manage personal link libraries. Along the way, he provides two utilities that streamline library management and explains how to shrink applications by linking stub modules.

DOS **65** **BIOS COMPRESSION** *By John E. Swanke*

The IBM PS/2 comes with a fat 128K ROM BIOS, but much of this memory space is unused under DOS. With BIOS compression you can reclaim some of this address space for DOS memory management.

ASM **69** **HASHING FUNCTIONS** *By John Otken*

John examines string hashing functions for fast table lookup and explains why a hash table, unlike your car's gas tank, should never be full. He also creates equivalent functions in assembly language for Windows 3 routines that manage handle-based dynamic storage and atoms (hashed strings).

COLUMNS

PJ FORUM **11** **COMPUTER STORES AND POPCORN KERNELS** *By Dave Thompson*

CONSULTING **14** **PROGRAMMER PRODUCTIVITY** *By William H. Roetzheim*

IMPRESSIONS **77** **LANGUAGE TRANSLATION AND 386 DEBUGGING** *By M. Steven Baker*

ARMCHAIR **80** **HARDWARE HELP** *By M. Steven Baker*

OFFRAMP **88** **HARDWARE DECISIONS** *By Hal W. Hardenbergh*

DEPARTMENTS

4 **EDITOR'S NOTE**

6 **LETTERS**

83 **PRESSROOM**

85 **PJ BILLBOARD**

72 **ADVERTISER INDEX**

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

EDITOR'S NOTE

To a hardware designer, problems are typically software-related; a software developer usually blames the hardware. When aging software and new releases encounter old and new hardware, incompatibilities are bound to arise. Attempts to enlarge DOS's 640K address space and the resultant complex layering of software make a bad situation even worse—network software, applications, and TSRs end up fighting for memory.

Major changes are occurring to software (Windows 3, DOS-extender applications, etc.), but the DOS hardware market appears to be stabilizing. The 386 was a big change from the AT's 80286, but the 486 adds little except speed and a built-in math chip. Although we now have XT, AT, MCA, and EISA buses, older DOS software appears to be living in harmony with these hardware environments. This can be attributed in large part to IBM's ROM BIOS, which hides hardware differences. Once you edge past DOS to OS/2 and Unix, hardware and software problems become more common. DOS 5, along with other protected-mode 386 software, is unfortunately awakening long-dormant bugs in many early Intel 386 chips. But these problems have an easy fix—replace your CPU.

WHO'S ON THE BUS AND WHO'S NOT

So far, machines based on IBM's Microchannel Architecture (MCA) appear to have delivered little in the way of enhanced performance over the original AT bus—now termed the Industry Standard Architecture (ISA) bus. As was widely predicted, MCA has been hampered by a more limited selection of option cards as well as by higher board costs. Meanwhile, newer 386/486 AT bus machines often sport adjustable bus speeds that range from 6 to 16 MHz, and newer disk controllers and network cards support these higher speeds. At speeds of 16 MHz, a fast ESDI or SCSI drive is limited at the drive, not at the bus.

EISA 386/486 machines have quietly arrived on the scene. The EISA boxes may be a bit overpriced now—EISA, like MCA, appears to offer little in additional performance unless it's used as a file or network server. At the market's high end, EISA may be sparring with MCA, but neither is generating much excitement or enthusiasm. And the convenience of hardware configuration via software—offered by both MCA and EISA—isn't preventing corporate or individual buyers from purchasing ISA machines. IBM has already used the MCA bus for its RISC 6000 workstations, so don't be surprised if Compaq and others choose the EISA bus for their upcoming RISC boxes.

LEGAL BEAGLES, A PLETHORA OF PIXELS, AND A FOND SO LONG

From the legal quarter, we have the Federal Trade Commission looking into unfair trade practices by Microsoft regarding Windows and OS/2. It's interesting that the companies most severely burned by OS/2's "fall from grace" are doing the most to aid the FTC investigations. Meanwhile, a variety of important software "look and feel" copyright cases are wending their arduous way through the legal system. Apparently, Intel does not have ownership of the numbers "386." Here at *PJ* we breathed a sigh of relief when we realized we could keep our trusty 800 line ([800] 234-0386).

Expect rampant hardware changes to continue in the PC graphics arena. The Edsun RAMDAC chip is in a good position to become a de facto standard on higher-end VGA boards. PC boards that support TIGA using TI's TMS34020 chips are coming down in price. The IBM XGA (extended graphics array) will soon be cloned, adding yet another graphics controller to support. And although we skipped the WGA, the YGA (You Guess Array) is next on board. Will the pixels never end!

After six years at *PJ*, our own stalwart and dapper sales director, Robin Tappan, is moving on to greater adventures. Robin's advice and good humor have benefited many software entrepreneurs. Those who know Robin know he's a one-of-a-kind type of guy. From all his friends at *PJ* and beyond, we wish him the very best.

Steven Baker

Programmer's Journal

Editor
M. Steven Baker

Managing Editor
Cecelia Hagen

Associate Editors
M. Elizabeth Hardinger, Deborah Healey

Contributing Editors
Michael Abrash, Brett Glass, Frank D. Greco, John Otken, Tom Swan

Founder
Greg Estes

Publisher
Elizabeth Oakley

Associate Publisher
Bobbi Sinyard

Sales & Marketing Director
Robert M. Tappan

Advertising Manager
Jay Moore

Advertising Coordinator
Bonnie Chin

Production & Art Director
Wanda Walker

Production Assistants
Margie Hanna, Nicole Harris

Subscriptions: Pamela Coleman
Office Staff: Tina Bates, Barbara Curtright, Jill Edwards, Sylvia Long, Randy Randolph, Bruce Searl

Business and Editorial Offices: ***Programmer's Journal,*** P.O. Box 70167, Eugene, OR 97401-0110, (503) 747-0800, (800) 234-0386, Fax: (503) 746-0071, BBS: See page 72

Subscriptions: 1 year (6 issues) $29.70

Change of Address: ***Programmer's Journal,*** PO Box 70387, Eugene, OR 97401-0121

© 1991 OAKLEY PUBLISHING COMPANY

Articles and letters in *Programmer's Journal* represent the views of the authors and not necessarily those of the publisher. Letters to the editor become the property of *Programmer's Journal* and may be edited for publication.

Note that all code and listings in *Programmer's Journal* appear "as is," and *Programmer's Journal* makes no guarantee or warranty of their quality, performance, accuracy, or suitability for any particular purpose. *Programmer's Journal* assumes no responsibility or liability to any person or entity with respect to any damage caused or alleged to be caused by their use.

Advertising: For ad rates, deadlines, and placement, call Jay Moore at 800-234-0386, or write to *Programmer's Journal,* P. O. Box 70167, Eugene,OR 97401-0110. *Programmer's Journal* is produced using the following software products: Ventura Publisher,® Aldus PC PageMaker,® and Corel Draw. The PostScript™ files are output on a Linotronic™ 200.

Programmer's Journal (ISSN 0747-5861) is published bimonthly by Oakley Publishing Company, 150 N. 4th Street, Springfield, OR 97477-5454. Second-class postage paid at Springfield, OR, and additional mailing offices. USPS # 000 293. **POSTMASTER:** Send address changes to *Programmer's Journal,* PO Box 70387, Eugene, OR 97401-0121.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The MetaWare Extended-DOS High C ® *Globally Optimizing Compiler* is for developing mission-critical applications that require:

- Compilation of large programs (one million+ lines)
- Ease of conversion of Microsoft "or other" code to 32-bits
- Support for the MetaWare 32-Bit Source-Level Debugger
- 100% ANSI Conformance
- Ease of migration to other development environments: DOS, UNIX System Vr3&4, SunOS, AIX, ...
- Global optimizations and Profiler for maximizing speed of code
- C++ as a future development language


```
if (environ == critical){
    survival = High_C;
```

The MetaWare Extended-DOS High C *Locally Optimizing Compiler* extends the capability of our industry-proven technology by:

- Exploiting the Power of AutoCAD® ADS and DESQview/X™
- Ease of conversion of Microsoft "or other" code to 32-bits
- Providing an execution Profiler that helps developers increase execution speed by identifying bottlenecks
- Supporting the MetaWare 32-Bit Source-Level Debugger

Professional software developers live and breathe in challenging development environments unknown to most programmers. As a matter of survival, these Pioneers depend on the precision, performance, and portability that only MetaWare delivers. The superior reliability and functionality of our quality compilers and tools allow these Explorers to navigate unfamiliar development terrain with confidence.

But even the best tools are not enough. You need a close, professional relationship with the veteran team that pioneered Extended-DOS compilers way back in 1986; this technical support is the best "Life support" available today. MetaWare now extends your development horizons with two choices in compiler technology. Both *Locally* and *Globally* optimized technologies support the new MetaWare 32-bit Source-Level Debugger.

To learn more about how MetaWare can help you survive in your environment, or to place an order, call now.

MetaWare®
INCORPORATED

Compiler Products for Professional Software Developers

Circle 3 on Reader Service Card

2161 Delaware Avenue • Santa Cruz, CA • 95060-5706 • 408/429-6382 • FAX 408/429-9273

MetaWare, High C, and Professional Pascal are registered trademarks of MetaWare Incorporated. Other names are trademarks of their respective companies.
© Copyright 1991 MetaWare Incorporated
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LETTERS

LANGUAGE-LUBBER SALUTE

Craig Banning, of Big Pine Key, Florida, was the first PJ *reader to decode the international code flags pictured on volume 9.2's nautical cover. The flags spelled out the names of five languages: C, Pascal, Fortran, ASM, and BASIC. Craig's quick response earns him a field promotion to Language-Lubber First Class, complete with a 21-bug salute and full regalia (a* PJ *T-shirt). Congratulations, Craig, and a yo-heave-ho!*

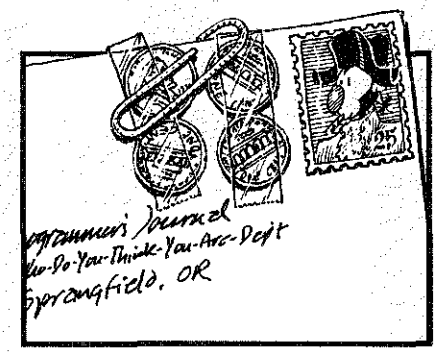


ILLUSTRATION BY JIM CARPENTER

A MORE CLEVER COMPILER

Like Clark A. Calkins ("Letters," *PJ* 9.1), I too have spent a fair amount of time disassembling the code that compilers generate. I work mostly with Microsoft compiled BASIC, and I can sympathize with Mr. Calkins's lament—a human will always be able to produce better assembly language code than a compiler. But Microsoft's latest BASIC 7 does a much better job than Turbo Pascal version 5.5.

Using his short code example, BASIC 7 produced this:

```
I = LastValue
    MOV  AX,LastValue
    MOV  I,AX
IF I > MaxValue THEN MaxValue = I
    CMP  AX,MaxValue
    JLE  Label
    MOV  MaxValue,AX
Label:
```

As you can see, BASIC's code size is less than three-fourths that of Turbo Pascal (15 bytes vs. 21), and in this case I do not see how the BASIC code could be further improved.

Note how BASIC cleverly reverses the sense of the comparison. Where the source tests whether **I** is greater than **MaxValue**, the compiler-generated code tests whether it is equal or less. This produces better code by minimizing the number of jumps added.

Ethan Winer
Crescent Software
West Redding, CT

INPUT FROM DISnDATa

In "Disassembler roundup" (*PJ* 9.2), Brett Glass compares several disassembly programs, including our DISnDATa for IBM PC compatibles.

Unfortunately, Mr. Glass does not fully understand how to use our disassembler. He writes that it "fails" in several cases where he feels he is not allowed to furnish enough information for DISnDATa to do its job.

He notes two places in which we notify users that more information may need to be input to disassemble the program completely, but he claims there are no means given to handle the translation of certain code elements into instructions. Our main design goal was to *minimize* the amount of user-supplied input needed to obtain a correct and complete disassembly.

We do indeed provide users with the ability to input any *additional* information that may be needed to perform a correct translation. As explained in the manual, DISnDATa separates instructions from data by examining each instruction in turn. When it encounters any indirect or "computed" JMP or CALL, DISnDATa asks the user to input the relative addresses to which control may be passed by this JMP/CALL. The trace then proceeds forward from each of these furnished addresses as it would from any other JMP/CALL.

After completing this trace from the standard entry point(s), DISnDATa asks the user whether there are any additional places that are entered solely from outside the program. This allows the user to input the locations (and segment register contents, if known) of all (nonstandard) interrupt routines and other instruction areas not reached from the standard entry points via JMPs/CALLs. The trace performs an analysis of each instruction and records the address and implied length of any memory operands. It then notes (as PROCs) the areas of the program reachable via each CALL instruction. Also, the current version (3.2.05+) will unpack LINK-packed modules.

We feel that if Mr. Glass had properly used the facilities of DISnDATa, he would have listed it as the most useful of the reviewed disassemblers. Who wouldn't prefer a disassembler that obtains most of the information directly from the file of the program being disassembled and that requires an absolute minimum of input from the user to produce an accurate disassembly?

C. W. Medlock
Pro/Am Software
Centerville, OH

BRETT GLASS RESPONDS: A disassembler that produces perfect output with no outside guidance would be highly desirable. However, during my tests, I discovered that even if I answered every question asked by DISnDATa as accurately as possible, it was unable to produce completely correct disassemblies of several of my test files. (All the others could, although the amount of "hinting" required to do this varied from program to program.) And, although DISnDATa's specialty is identifying and reporting on data, there was no way to tell the disassembler about data structures (such as the character/vector table in ANSI.SYS) so that a clear, formatted listing could be produced. None of the other programs had this restriction.

With DISnDATa, I could not find a series of responses that would trigger complete and correct disassemblies of ANSI.SYS (from IBM's PC-DOS 3.3), TORTURE.EXE (source included in the article), and XMSDUMP.EXE (source on p. 73, PJ *9.2).*

PITY THE POOR POPPIES

I really enjoyed Hal Hardenbergh's column in *PJ* 8.4. I agree with it 100%. However (you were waiting for this), I must point out that B-2 bombers don't bomb poppy plants in Colombia or Peru. There aren't any there.

Hal is correct in saying that Congress could change any law they want, any time they want. They never will. They all have the same disease that my senator has—they know more about what is best for me than I do!

Dan S. Sullivan
Lowell, MA

OFFRAMP OFF BASE

I do not feel that Hal Hardenbergh is well informed on C and its uses in mathematics and compiler technology (*PJ* 8.6). Pre-ANSI C compilers might be considered "hostile to floating point." ANSI C compilers, however, are no more hostile than most computer languages.

Many scientists requested that the

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

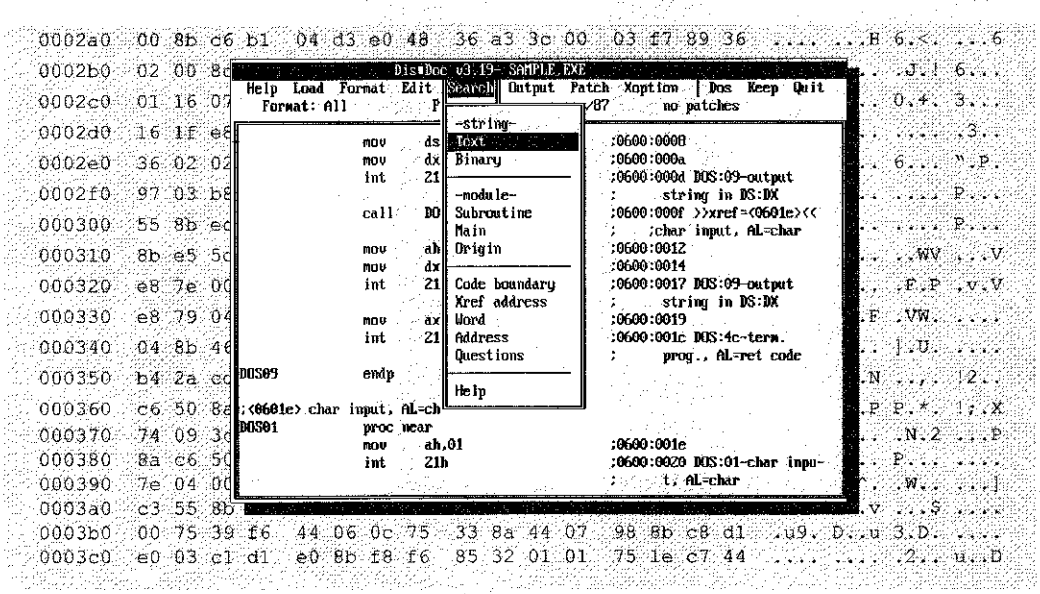
NEW VERSION

No Source?
No Problem.

Automatic BIOS, PSP, MS-DOS interrupt labeling built-in

Batch and Interactive modes

Interactively locates code/data boundaries



8086 to 80486 instruction sets supported

Interactive cross-referencing

Creates listing only after disassembly is perfect

Dis • Doc Professional™

New **Dis • Doc** *Professional* is your **dual-mode** key to any DOS source code. It works in batch and interactive modes simultaneously, allowing you to generate the core information of even the most complex programs **fast**...and modify them even faster. Most programs will come apart in just two minutes. Imagine what you can do with a tool this powerful! **Dis • Doc** sifts through programs **eight times** for guaranteed accuracy. When code gets mixed up with data, our toolbox comes to the rescue with **smart** search and **easy** edit utilities.

Warning: Dis • Doc *Professional* may change the way you work **forever**.

Programmers who used to shy away from fixing outmoded programs with no source code are going to discover a **valuable** new talent: the ability to modify and revise codes that would cost way too much to start over (it's a programming manager's dream). Save your employer huge new-programming fees and enhance your marketability. **Dis • Doc** is so easy to learn, you'll be a high-dollar hero in no time!

Dis • Doc *Professional* is an amazing new teaching tool. Learn how programs work...take them apart and see the writing techniques that top pros use. Use it to assist in debugging. Hunt down viruses and write killers. **Dis • Doc** can save you **years** of frustration, and it only costs $249.95.

COMPARE FOR YOURSELF!

	Dis•Doc™	Sourcer™
Speed of disassembly (bytes/sec)	***822***	*120*
Accuracy (code/data separation*)	***99.9%***	*99.9%*
Unlimited file size	***YES***	*NO*
BIOS labeling	***YES***	*XTRA $$*
BIOS/PSP file labeling	***YES***	*NO*
Unpacks EXE files	***YES***	*XTRA $$*
Patcher	***YES***	*NO*
Batch mode	***YES***	*YES*
Interactive mode	***YES***	*NO*
Time it takes to customize a label	***5 sec.***	*2-5 min.*
Time it takes to change data to code	***5 sec.***	*2-5 min.*

*at eight passes based on a 9040 byte EXE file picked at random

To order your **Dis • Doc Professional**™ disassembler toolkit or our $3 demo disk, simply call:

1-800-336-1961 in US
1-203-953-0236 outside US
or send check or money order for **$249.95** plus $5 s&h to:

RJSwantek, Inc.
178 Brookside Road
Newington, CT 06111-1320 USA

For more information call 1-203-953-0236 • FAX available • MasterCard & Visa • Shipped Immediately Via UPS Blue inside US.

Sourcer is a registered trademark of V Communications, Inc.

Circle 4 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ANSI committee establish rules for handling float types (Cray Research even requested that language features be added to support specific processors). The rules that the ANSI C standard set in place regarding promotion of float eliminated most problems (e.g., functions declared with float parameters via prototypes receive float arguments on the stack).

Computer languages are an abstraction of human thought. The compiler's job is to create machine instructions that execute efficiently on the host processor. Hardenbergh is way off base in claiming that C (or any computer language) is not suited for a particular processor architecture. It is up to the compiler to create acceptable machine code for a particular architecture (a computer language may include hints that help the compiler generate code). RISC-based C compilers are good examples of compiler technology that has successfully adapted to a new processor architecture.

Creating computer languages for a specific processor architecture defeats the purpose of high-level computer languages. Computer languages should evolve to mirror human thought more clearly, not to adapt to specific hardware. Let the compilers do the work!

Phillip Lindsay
Anaheim, CA

SEEING IS BELIEVING

Hal Hardenbergh claimed (*PJ* 8.6) that the C language is spectacularly ill-suited for the latest, fastest microprocessors. What is really ill-suited is Hardenbergh's tendency to comment on a subject that he admits he knows very little about.

The ability to handle floating-point numbers is solely the property of the implementation, not of the language. Hardenbergh claims that C is tied to current compiler technology and therefore cannot make use of command pipelining and other advanced technologies. Hogwash. The only thing that is tied in C is the language specification, which is (fortunately) standardized and independent of the processor architecture on which pipelining and other advanced technologies are based. The language specification has no business in this vendor-specific hardware area.

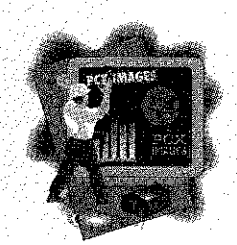
The implementation can make use of advanced technologies, and there is *no* inherent limitation imposed by the language specification. For instance, the VAX-C compiler for VMS handles the decomposition of source code for multiprocessor environments, breaking down iterative loops that can be executed simultaneously on more than one processor. The VAX-C compiler also optimizes for hardware array processors that may be present in the system at execution time.

C is particularly *well*-suited to the new microprocessors because it provides them with a complete operating system (Unix). Second- and third-party applications can be made available very quickly on new processor architectures. Only a fool would choose to write a new operating system for each new microprocessor being introduced.

Hardenbergh should stick to the discussion of BASIC compilers for the Atari ST. As he put it, he "knows a little about this."

Todd Merriman
Ball Ground, GA

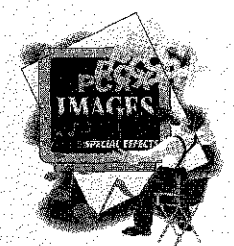
ADD THE POWER OF GRAPHICS TO YOUR PROGRAMS WITH GENUS



PCX PROGRAMMER'S TOOLKIT 4.0

Finally, a programmer's toolkit that helps you incorporate graphics into your programs quickly and easily. The PCX Programmer's Toolkit includes over 75 routines to display, save, scale, and print PCX bitmapped graphics from almost any program.

..... ONLY $249

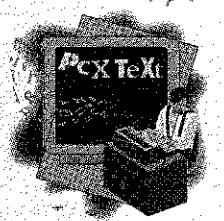


PCX EFFECTS*

Add spectacular special effects (F/X) to your programs, without being trapped in someone else's editor or slideshow program. Wipe, split, crush, slide, sand, drip, diagonal, spiral, random, or explode your graphics.

..... ONLY $99

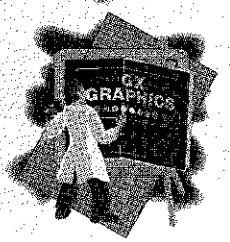
* Requires PCX Toolkit



PCX TEXT

Now you can display blazing bitmapped text in any graphics mode as simply as text mode. With PCX Text, you can display your textual information right along with your graphical data.

..... ONLY $149



GX GRAPHICS

A complete graphics library supporting all graphics primitives. Use the GX Graphics toolkit instead of the BGI or MS libraries and make your program faster, smaller, and more portable across compilers – while supporting more video modes.

..... ONLY $199

FOR THE GENUS GX DEVELOPMENT SERIES CATALOG, CALL TOLL-FREE:

1-800-227-0918

VISA, MASTERCARD, AMERICAN EXPRESS, CHECK, C.O.D., PURCHASE ORDER

G·E·N·U·S MICROPROGRAMMING

11315 MEADOW LAKE • HOUSTON, TX 77077 • (713) 870-0737
FAX (713) 870-0288 • BBS (713) 266-9362

Circle 5 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

SOME ASSEMBLY REQUIRED

I'm a *PJ* fan. I look forward to articles that involve ASM code or relate to the inner workings of DOS or the BIOS. I realize that in this age of HLLs, GUIs, mice-addicted users, and OOPs-oriented developers, that attitude isn't too stylish. But I see three principal reasons for the use of ASM code: to write little goodies to enhance our HLL code; to do embedded code or device-oriented functions; or to do our own thing!

The last reason, of course, is the one that poses the real challenge and also seems to be the one that is least understood by ASM popularizers. Apparently, few of us realize that only by using ASM code can we create any kind of software we can imagine.

ASM's redeeming features include more than its improvement in execution time or reduction in memory usage. Only with ASM can we design any off-the-wall data structure that our Abstract Data-Type minds can conceive.

The style of ASM coding one uses and the weight assigned to individual dicta will vary—should vary—depending on which star one is following at the moment. That means most of the rules about how to write assembly language code need to be rewritten.

There is a real need for a more intelligent way to write assembly language. ASM's only drawback is the time required to develop an application with it, which, of course, is because of the way it is taught, implemented, and practiced. ASM is not innately complex or ambiguous, but the assembler and its documentation can make it seem that way. If it is to remain pure ASM code, there is no reason not to let assembly code slide up and down the scale of abstraction.

The overriding problem of designing and implementing a new assembler language is maintaining backward compatibility. We mustn't lose all the code we've already written, but we can't allow that to hamper our search for the Holy Grail (a better ASM world for all of us).

Mark Rhyner
Chicago, IL

ERRATUM: The wrong telephone number was published for Richard Riehle in his article in *PJ* 9.2. The correct number is (415) 858-1551.

YOU OUGHT TO BE IN
☆ REPRINTS ☆

You're the picture of success.

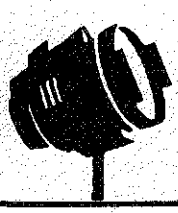
And **PJ** reprints are a great way to showcase your four-star performance. Reprints help you:

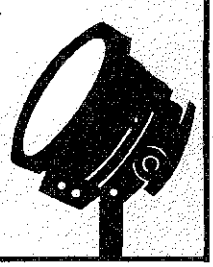
- ☆ Enhance your trade show presentations
- ☆ Promote your products and services
- ☆ Increase your marketplace presence
- ☆ Optimize your product training sessions

PROGRAMMER'S JOURNAL offers full-color or black-and-white reprints.

PJ reprints keep you in the spotlight.

Call us today at 800-234-0386.





Debug C

What's *your* problem?

- *program hanging?*
- *memory overwrites?*
- *operating system overwrites?*
- *corrupted heap?*
- *memory leaks?*
- *dangling pointers?*
- *open file pointers?*
- *invalid calls to free()?*

If your code has a bug that's making *you* sick, *call the doctor*—**Dr. MD, the run time memory debugger for C!** Sure, you *could* find it yourself, but how much would it cost you in time, money and aggravation?

Dr. MD is the **easy and cost effective** way to cure your memory corruption ills. **Dr. MD** gives a **complete diagnosis** including the source file and line number of your bug.

- *includes source code*
- *non obtrusive*
- *easy to customize*
- *hardware independent (386 not required)*
- *operating system independent (DOS, UNIX, OS/2...)*
- *small library*

Start your code on the road to recovery—order your copy today!

DR. MD™

NEW version 2.0
$129.00 plus shipping

RSS Engineering
Custom and Contract
Software Solutions

2683 N. Taft
Loveland, CO 80538 USA
(303) 669-3300
FAX (303) 669-3354

MasterCard VISA

RSS Engineering
a division of
Remote Switch Systems, Inc.

Circle 6 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Conquer OS/2 !!

Now OS/2 can respond to your every whim

Introducing:

Assistant

PM Assistant is the **first** and **only** Automation and Customization Utility for OS/2 Presentation Manager

- Automate repetitive tasks with **mouse and keyboard input macros.**
- Play keystrokes to any PM application or full-screen session.
- Start, position, and initialize your PM applications **automatically.**
- Switch to and position PM applications and full-screen sessions with user-defined **hot-keys.**
- Execute any macro from within any PM application or full-screen session with user-defined **hot-keys.**

- Set, view, and change *any* program's **OS/2 process priority.**
- Assign **accelerator keys** and rearrange the menu commands of *any* PM application.
- Speed dialog box interaction with Speed Button.
- Prevent damage to your monitor with **Screen Saver.**
- Protect your computer from unauthorized access with **Password Protection.**
- Create turn-key procedures or demonstrations of your PM programs with the **demonstration utility.**

Now, all your favorite tools in Dos and Windows are available for Presentation Manager along with features unique to OS/2. PM Assistant comes with on-line help, an unconditional satisfaction guarantee, and a **free program** - DiskWatch, a disk free-space monitor. PM Assistant runs on OS/2 versions 1.10 or greater, and is now available at a special introductory price:

PM Assistant.....$125.00

Take control of OS/2, order today!

1-800-745-7757

Utilis Advanced Tools Now

Trademarks are property of their respective holders.

P.O. Box 15905
Seattle, WA 98115
(206) 467-4025

Circle 7 on Reader Service Card

COMPUTER LANGUAGE PRODUCTIVITY AWARD 1990

INSTALL.BAT

First impressions last.

INSTALL 3.0 gives your software product a professional introduction.

Installations that rely on batch files look amateurish and have limited error handling. INSTALL 3.0 creates an elegant installation procedure that will increase user's confidence in your software product.

INSTALL 3.0 comes with many samples you can modify and use immediately; no programming usually needed. All available RAM is used for fast file transfers.

- Features very sophisticated error-handling
- Does high performance data compression
- Can safely and reliably modify system files
- Allows selective installation of parts of your program
- Complete source code included
- No royalty charges
- Free technical support

INSTALL 3.0 detects monitors, DOS versions, coprocessors, etc. to create machine-specific installations without asking questions.

INSTALL 3.0 has been used for over four years by some of the biggest (and smallest!) names in the industry, in the U.S. and abroad, to install millions of copies of programs. Make sure your software product makes a good first impression with INSTALL 3.0.

KNOWLEDGE DYNAMICS CORP.
Highway Contract 4, Box 185-H
Canyon Lake, TX 78133-3508
MasterCard/VISA/COD/POs welcome

$249.95 + shipping
Money back guarantee
Sales 800-331-2783 Ext 0196
Internat'l 512-964-3994
24 hr FAX 512-964-3958
24 hr BBS 512-964-3929

Circle 8 on Reader Service Card

OPUS SOFTWARE

"Without a doubt, OPUS Make is the hottest make utility on the market"
Tom Swan, PCWorld

OPUS MAKE

For professional programmers, OPUS Make is the superior choice in a make utility. It is the fastest most full-featured make utility there is. Features include:

- **DOS version uses only 3K memory!**
- **Multiple directory support.**
- **Supports: Polytron PVCS™**
  Burton Systems TLIB™
  Microsoft LIB™
  SLR Systems OPTLIB™
- **Generates automatic response files of unlimited length for LINK and LIB.**
- **OPUS MKMF included: An automatic dependency generator that fully understands C preprocessor directives.**

Both DOS and OS/2 executables for only: $129 MC, Visa, COD, or PO. 60 day MBG!

Call 1-800-248-OPUS
International: 415-904-7101
1032 Irving Street, Suite 439, San Francisco, CA 94122
Trademarks owned by their respective companies

Circle 9 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Computer stores and popcorn kernels

By Dave Thompson

Not everything related to computers is fun and frivolity: sometimes I have to go to a computer store and actually buy something. There's one place where I do most of my shopping, most of my browsing, most of my drooling, and most of my standing around watching others drool.

I like the place because the staff lets me make snide remarks about their boss (my brother) and because they usually have a large bag of stale popcorn (the really good stuff, with palm oil, extra salt, and artificial color). So lazy afternoons often find me and the boys standing around the popcorn sharing computer stories and spitting hard kernels. The paying customers have to entertain themselves (unless, of course, they have their own computer stories and like cold popcorn).

Anyway, between my brother and that popcorn it had been a while since I'd gotten out to see what the other computer stores were up to. But then I started nosing around. Here in Bend, we have more computer shops than grocery stores (not necessarily better, just more). Computer stores seem to sprout in the fall and die off by spring, a reversal of the natural order.

One had lots of used stuff, thoroughly used. Like Commodore 64s, TI 99s, Radio Shack model 1s. They even had ham radio transceivers.

Another had just two computers in stock. When I started pricing their software programs, I found they were cheap—really cheap—like three for $25. I recognized one package, a prehistoric copy of Wordstar. The only printer it listed was "Chisel." The rest of the software, I'd never heard of.

The packages, it turned out, had been remaindered. Programs get treated like books and cassette tapes: if

ILLUSTRATION BY JIM CARPENTER

it doesn't sell or if copies are left over when a new version comes out, distributors sometimes ship the remains to remainder houses. Computer stores buy them cheap and sell them for whatever they can get. Remaindered software is wonderful if you know what you're getting, but don't figure on getting any support. You shouldn't pay more than 10 or 20% of the original list price.

A SAD STORY OF MY EJECTION

I hadn't been in another Bend store, Computerland, since the time I was physically ejected. I needed the latest version of Microsoft's assembler to finish a project, and they were the only place in town that had a copy (and the only place in town that usually charges full list price). I'd called ahead, so I knew they had it.

It was about closing time, and I was the only customer. I asked to purchase the assembler. The guy just stared at me.

"What kind of system?" he asked.

"Something I've cobbled together."

"Floppies?" he asked.

"Well, yeah, of course."

"Hard drive?"

"Uh, sure."

"Then the assembler won't work for you." He smiled.

"Why not?"

"It doesn't work on hard drives." His smile broadened.

"It doesn't care whether I have a hard drive," I said. "It's a silly assembler."

"It doesn't work on hard drives," he repeated.

"Look, the operating system takes care of hard drives, floppy drives, RAM drives and all that. Just sell me the assembler."

"I've had too many years of computer experience to sell you anything," he said, beaming.

"Let me guess. You've just been paroled after seven years, alone, in a small, windowless room writing COBOL for an IBM mainframe," I said.

He turned crimson, grabbed me by the collar, escorted me out, and locked the door.

(I bet he'll never offer to share his popcorn with me.)

NOT HUCKLEBERRY FINN

You can also find lots of computer stuff in bookstores. In fact, there's more computer knowledge at B Dalton's than in an entire computer magazine editor. (And the information at B Dalton's might even be useful.)

Now you may be wondering where all that knowledge comes from—or whether anyone cares. First, you have to understand that very little information is ever actually new, but that never stops the barrage of books.

For instance, the latest dBASE IV book is just an edited version of the dBASE III book. The dBASE III book was an edited version of dBASE II, and dBASE II came from dBASE I. dBASE I came from the public domain—which is why Ashton-Tate will spend something like $8 million to rewrite the law or rent a congressional rep or do whatever they have to do to get copyright protection again.

Anyway, I was looking through my bookshelf for a good book to tell you about. All I can really tell you is that there aren't many books I wish I'd written.

Sure, I'd like to have a few C++ titles out there, such as *C++ for the Masses, C Is No Object, but C++ Is, Oh Say Can*

Dave Thompson is a former hardware/software engineer at Tektronix and the past editor of the prehistoric programming magazine Micro Cornucopia. *He can be reached at P.O. Box 223, Bend, OR 97709.*

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

You C++, or *Small-Business Owners' Guide to Writing Spreadsheets in C++*.

Turbo Modula-2 Users Guide could have been a biggie, but Borland never released its Modula-2. *QuickPascal Programming* has no doubt sold more copies than Turbo Modula-2—maybe five more. Rumor has it that Microsoft has assigned a half-time programmer to update QuickPascal, so in 10 or 15 years we could see this book's first printing. Or I could have written *Turbo Prolog Primer* or one of the plethora of guides to OS/2. (I understand OS/2 was going to be an operating system.)

PEN-BASED COMPUTING

This is one of those ideas that could go nuts or could die a quiet death. I know there are a hundred zillion programmers who'd love to stop hunting and pecking and go back to scribbling.

As you've no doubt heard, Go Corporation is showing early versions of its pen-based system to the press. (Talk about some guys who wouldn't dream of giving up *their* Crayolas.) It sounds interesting enough—even the freehand sketching sounds fun.

But the problems are obvious. My handwriting is about as legible and consistent as your average MD's, and this package doesn't even pretend to understand handwriting. Of course, character recognition will have to improve dramatically if this operating system becomes popular.

I remember the high-res bar codes that *Dr. Dobb's* used to print. Several pages of listings would fit on a single page of these stripes, and the code was claimed to be error free. Each reader was supposed to purchase an $80 scanner, and—presto, bingo—out would come instant source code. The company (I can't remember its name) tried to get me to run the bars in *Micro C*. Almost had me doing it, too. However, *Dr. Dobb's* dropped it at about that point because folks weren't purchasing scanners.

Pens, however, will come. Whether pen-based computing will soon be a Go, I don't know.

EYE-OPENING EDUCATION

I'm taking a class in fiction writing at the local community college, and one thing that's really hit me is the cost of books and supplies. Wow! Would you believe $30 for a paperback. Plus, I have to make 20 copies of my work for my classmates' critique. (Talk about getting no respect—you should hear those vultures.)

If we all had laptops, we could simply cable-up our machines and share stories. And the school could order one electronic copy of a textbook. Then for, say, a $10 fee, we'd get a plug-in copy. Hard to limit circulation, I know, but we'd save a lot of trees, landfills, and rivers. Thirty-five percent of landfill waste is paper.

Oxymoron of the month: Technical literature. PJ

Data Junction® converts data files!

Any ASCII • dBASE • R:BASE • Lotus 1-2-3
Excel • Paradox • Informix • Q&A • Oracle • Clarion
DataFlex • askSam • SC4 • Xdb • DataEase • Symphony
ACT! • Platinum • DAC • ACCPAC+ and other apps
Btrieve • c-tree • EBCDIC • Packed & Zoned decimal
IBM/Mips/Sparc/Cray/Vax/Intel/68000 Binaries & Floats

Filter Your Files While Converting!
- *Menu driven — No programming*
- *Select & rearrange records/fields/chars*
- *Instant scripting for batch converts*
- *Written in C — Ideal for OEM apps*

Clean Up & Massage Your Data!
- *Merge & split fields; Name/address parsing*
- *Search/replace and case translation by column*
- *Edit/enter corrections — Build custom data types*

1990 DATA BASED ADVISOR The Database Management Systems Magazine Readers Choice Award

"... an invaluable integration tool." —Byte
"... more than worth its price." —PC Magazine
"... a surefire winner." —Computer Language

Standard $99— Converts dBASE, ASCII, 1-2-3, Q&A, Enable, DIF, askSam, & merge files for WordPerfect, Word & Wordstar
Professional $199— Standard formats plus Binary/EBCDIC, SC4, SYLK, Paradox, R:BASE, Clarion, Excel, DataEase, DataFlex, c-tree, Btrieve
Advanced $299— Standard & Prof. formats + Informix, Xdb, Oracle, C-ISAM

Call **800/444-1945** or 512/482-0824 • FAX 512/482-0976
Also available for LANs, Xenix & Unix. MC/VISA • AMEX • COD • Corp. PO.
Tools & Techniques Inc • 1620 W 12 • Austin TX 78703

Circle 10 on Reader Service Card

The Programmer's Corner
— A Source Code Repository —

If you are reading this magazine, then you appreciate the art of programming. One of the most important tools for a software craftsman is example code. To meet this need, we provide **more than 9000 files** (over one gigabyte) of shareware and public domain source code and programs. We cover all popular languages including C, Pascal, and Basic. TPC even has hard to find source code such as OS/2 PM C source, C++ source, and Turbo Pascal 5.5 OOP source. Source code from this issue of *Programmer's Journal* is also available. New code is added daily, all in the interest of helping the professional programmer.

So call today: (301) 596-1180 300-9600 Baud
(301) 995-3744 24 Hours-A-Day
(301) 621-4636
Only $50 for a one year subscription

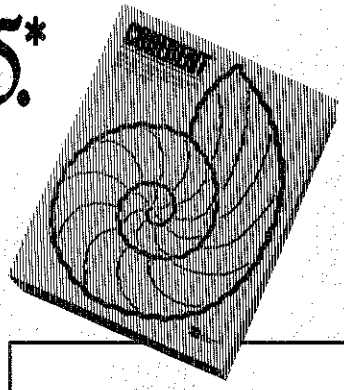
— TPC uses FastComm 9600 V.32 Modems —

TPC **The Programmer's Corner**
5430 F Lynx Lane, Suite 224
Columbia, MD 21044

Circle 11 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Everything You Ever Wanted In UNIX. And Less. $99.95.*



NEW COHERENT RELEASE 3.1
NOW WITH…
–elvis: vi editor clone
–SCSI (Adaptec AHA 154x series and more on the way.) and ESDI support
–UUCP Bulletin Board System
–RAM disk support
–And much, much more!

OVER 10,000 SATISFIED USERS!

OK. We know it's hard to believe. So just consider this. Coherent™ is a virtual clone of UNIX. But it was developed independently by Mark Williams Company. Which means we don't pay hundreds of dollars per copy in licensing fees.

What's more, Coherent embodies the original tenet of UNIX: small is beautiful. This simple fact leads to a whole host of both cost and performance advantages for Coherent. So read on, because there's a lot more to Coherent than its price.

SMALLER, FASTER…BETTER.

Everybody appreciates a good deal. But what is it that makes small so great?

For one thing, Coherent gives you UNIX capabilities on a machine you can actually afford. Requiring only 10 megabytes of disk space, Coherent can reside with DOS. So you can keep all your DOS applications and move up to Coherent. You can also have it running faster, learn it faster and get faster overall performance. All because Coherent is small. Sounds beautiful, doesn't it?

But small wouldn't be so great if it didn't do the job it was meant to do.

LESS IS MORE!	Coherent For the IBM-PC/AT and compatible 286 or 386 based machines.	Santa Cruz Operation's XENIX 286, Version 2.3.2
No. of Manuals	1	8
No. of Disks	4	21
Kernel Size	64K	198K
Install Time	20-30 min.	3-4 hours
Suggested Disk Space	10 meg	30 meg
Min. Memory Required	640K	1-2 meg
Performance*	38.7 sec	100.3 sec
Price	$99.95	$1495.00

*Byte Execl benchmark, 1000 iterations on 20 MHZ 386. Hardware requirements: 1.2 meg 5¼" or 1.4 meg 3½" floppy, and hard disk. Does not run on Microchannel machines.

EVERYTHING UNIX WAS MEANT TO DO.

Like the original UNIX, Coherent is a powerful multi-user, multi-tasking development system. With a complete UNIX-compatible kernel which makes a vast world of UNIX software available including over a gigabyte of public domain software.

Coherent also comes with Lex and Yacc, a complete C compiler and a full set of nearly 200 UNIX commands including text processing, program development, administrative and maintenance commands plus UUCP.

CRITICS AGREE: IT'S AN INCREDIBLE VALUE!

"Mark Williams Co. seems to have mastered the art of illusion; Coherent comes so fully qualified as a UNIX clone, you find yourself thinking 'I can't believe it's not UNIX.' "
–Sean Fulton, UNIX Today!, November 26, 1990

"…(Coherent) may be the best thing that has happened to UNIX yet."
–William Zachmann, PC Week, November 5, 1990

"If you want to come as close as you can to real UNIX for a low price, COHERENT can't be beat."
–Warren Keuffel, Computer Language Magazine, November 1990

"If you want a UNIX-like development and learning system for less than $100…I don't see how you can go wrong with Coherent."
–David Fiedler, BYTE Magazine, November 1990

EXPERIENCE, SUPPORT AND A 60-DAY MONEY BACK GUARANTEE.

Wondering how something as good as Coherent could come from nowhere? Well it didn't. It came from Mark Williams Company, people who've developed C compilers for DEC, Intel, Wang and thousands of professional programmers.

We make all this experience available to users through complete technical support via telephone. And from the original system developers, too!

Yes, we know $99.95 may still be hard to believe. But we've made it fool-proof to find out for yourself. With a 60-day money-back no-hassles guarantee.

You have to be more than just a little curious about Coherent by now. So why not just do it? Pick up that phone and order today.

You'll be on your way to having everything you ever wanted in UNIX. And for a lot less than you ever expected.

1-800-MARK WMS
(1-800-627-5967 or 1-708-291-6700)
FAX: 1-708-291-6750
60-DAY MONEY BACK GUARANTEE!

Mark Williams Company
60 Revere Drive
Northbrook, IL 60062

*Plus shipping and handling. Coherent is a trademark of Mark Williams Company. UNIX is a trademark of AT&T. XENIX is a trademark of Microsoft.

Contact this advertiser directly.

PRODUCT INFO @ RETROBSD.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Programmer productivity

By William H. Roetzheim

Most managers agree that improving the productivity of their company's technical staff is important, but how important is it? My experience shows that on a large project, the cost difference between a single highly productive programmer and a less-productive counterpart can amount to a factor of five. That kind of money makes it worthwhile to investigate the nature of programmer productivity and how it can be improved.

BETTER, BEST, AND BLAH

Although the actual numbers will vary, we can get a feeling for the magnitude of the difference in productivity using some specific numbers from my experience. Let's begin by dividing programmers into three categories. **Average programmers** make up 80% of your programming staff and are often described as "competent" and "professional." In the environments I have worked in, average programmers tend to produce about 500 lines of code per month. A typical average programmer also introduces 15 coding errors per 1,000 lines of code (we won't deal with specification or design errors in this article). Average programmers could cost your company close to $5,000 per staff month (including wages, benefits, and overhead).

Super programmers may constitute 10% of your programming staff. Words like "guru" and "indispensable" come to mind. Super programmers will cost a bit more—perhaps $8,000 per month—but they also consistently produce 2,000 lines of code per month. In addition, a typical super programmer introduces only three coding errors per 1,000 lines of code.

ILLUSTRATION BY JIM CARPENTER

Ineffective programmers represent about the same percentage—10%—as super programmers; they're seen as "trying hard" and "cooperative." Ineffective programmers cost about the same as average programmers ($5,000 per staff month) and, surprisingly, produce about the same amount of code per month (500 lines). The difference lies in coding errors; ineffective programmers typically introduce a whopping 80 errors per 1,000 lines of code.

YOU CAN PAY ME NOW . . .

How much does it cost to repair a coding error? Two staff days to find and correct a coding error is a reasonable approximation. At the average programmer wage, this amounts to $250 per error. For a look at the cost per thousand lines of code for each of the different categories of programmer, see Table 1.

For a project consisting of 50,000 lines of code, the super programmer could do the work for $237,500, the average programmer would cost $657,500, and the ineffective programmer (with average programmers to fix the errors) would cost $1,500,000! Every month worked by an ineffective programmer requires four staff months of average programmer time to repair the bugs. Ineffective programmers actually introduce negative productivity into a project team.

Improving the quality of your work force brings significant potential cost savings. Managers have seen similar saving potentials in various forms of manufacturing processes; workers using modern, automated tools are more productive—by an order of magnitude or more—than their traditional counterparts. The obvious parallel is to develop faster, better (more modern) programming tools that allow average programmers to meet or beat the performance of the super programmers.

The current popularity of CASE tools is an example of this attempt. Unfortunately, soon after newer and better tools are introduced, we often observe that although overall productivity may go up (or down!), the relative gap between super programmers and average programmers remains constant. Power tools don't prevent mistakes, and they often aggravate an error's impact (it's easier to cut off your finger with a power saw than with a handsaw).

Another way to improve productivity is to hire only super programmers. Although logical, this alternative is impossible to implement because *there are not enough super programmers to go around.* Average (and ineffective) programmers are easy to hire. Super programmers are almost never looking for a job, and smart companies will pay the price to keep them. A more realistic solution is to keep your super programmers at all costs, work to turn your average programmers into super programmers, and identify and weed out the ineffective programmers.

USING GROWTH POTENTIAL

A "grow your own" approach to developing super programmers can work—I have found that, under the right circumstances, about one-third of all average programmers have the potential to become super programmers. To succeed at bringing out this potential, you need to consider your company's physical

William H. Roetzheim is a senior associate with Booz, Allen & Hamilton, and a noted author and speaker on software engineering topics. This article is based, with permission, on the author's book Structured Computer Project Management, *Prentice-Hall, 1988. Bill can be contacted at Booz, Allen & Hamilton, Inc., Suite 109, 4025 Hancock Street, San Diego, CA 92110; (619) 223-5681.*

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C Programmers Know What Works

Get a Good Start

When you find something that works, you keep using it. When that something is versatile and reliable, you use it a lot. C is that kind of tool. *SuperFunctions* is that kind of library.

Expanded Memory. Pull-Down Menus. Mouse for text and graphics. A Critical Error Handler. These are things you'll need. Maybe you'll be writing a package to do program trading on the Stock Exchange. Maybe your program must decode satellite data on atmospheric ozone.

Whatever it does, every program needs some basics. Like memory management. Like a user interface. You could write these functions yourself, but you don't need to, because they are already in *SuperFunctions*. Don't do what has already been done. Get *SuperFunctions* and get on with your real work!

Expanded Memory

Take along all the luggage you want. If your programs need more than 640k of memory, you can have it with *SuperFunctions*. Sense installed expanded memory. Use the **LIM 4.0** spec to map in pages and pages of expanded memory as data segments. Up to 32 megabytes. Effectively, **memory limitations will disappear** for you.

Critical Error Handler

Get some insurance. The days of letting DOS write "Abort, Retry, Ignore" right on top of your beautiful interface are gone. *SuperFunctions* gives you an elegant means to handle open drive doors, disk, and other critical errors. When a critical error occurs, your program can **intercept the error and recover** using the function in this library.

Logical Windows

Simply a necessity. Every beginner book on C has a chapter on text windows, and that's nice -- for beginners. You'll need more than that. *SuperFunctions* gives you more than 250 logical windows. You can move any one of them to the top with a single function call. You can delete a window and update the screen with a single function call. Write to a window which isn't even on the screen or one that is partially covered by another window. Keyboard idle functions keep the screen alive even when the user is not responding. Write to the window using lots of different text attributes, different colors, different intensities, word wrap on or off, etc. With *SuperFunctions*, it's all simple, automatic, worry free, and fast.

Menus

Use a good map. People use menus because they're simple, useful, and they work. *SuperFunctions* gives you menus that are **simple, useful, workable, and powerful.** One easy to use function determines how big the menu windows should be, displays them, deletes them when finished, accepts first-letter key matches, and indicates the selection to the calling program.

Mouse

Your programs need to know how to control a mouse. *SuperFunctions* gives you that power. **Mouse detection, click detection, position detection.** Complete interface for graphic and text modes.

Extras

SuperFunctions is loaded with extras. But we're running out of room here. We'll just mention that it has **CMOS Timer and Alarm, DOS file functions, advanced time and date, command line parsing,** and more.

Nuts and Bolts

Greenleaf's *SuperFunctions* is a nuts and bolts library with functions you'll really need. It will help you go a long way, and if necessary, you can actually get inside it and fine tune things yourself, because we give you all the source code. **FREE.** So have a ball.

FREE Bumper Sticker

Sure, a bumper sticker is silly. We know that. But it's also kind of fun, and when it's done right, so is programming. Find out more about our libraries and get a FREE bumper sticker, call us.

1-800-523-9830

Ask for extension #200. Also, don't forget to ask about our Gold Support Program.

FAX: (214) 248-7830

SuperFunctions™...$299.

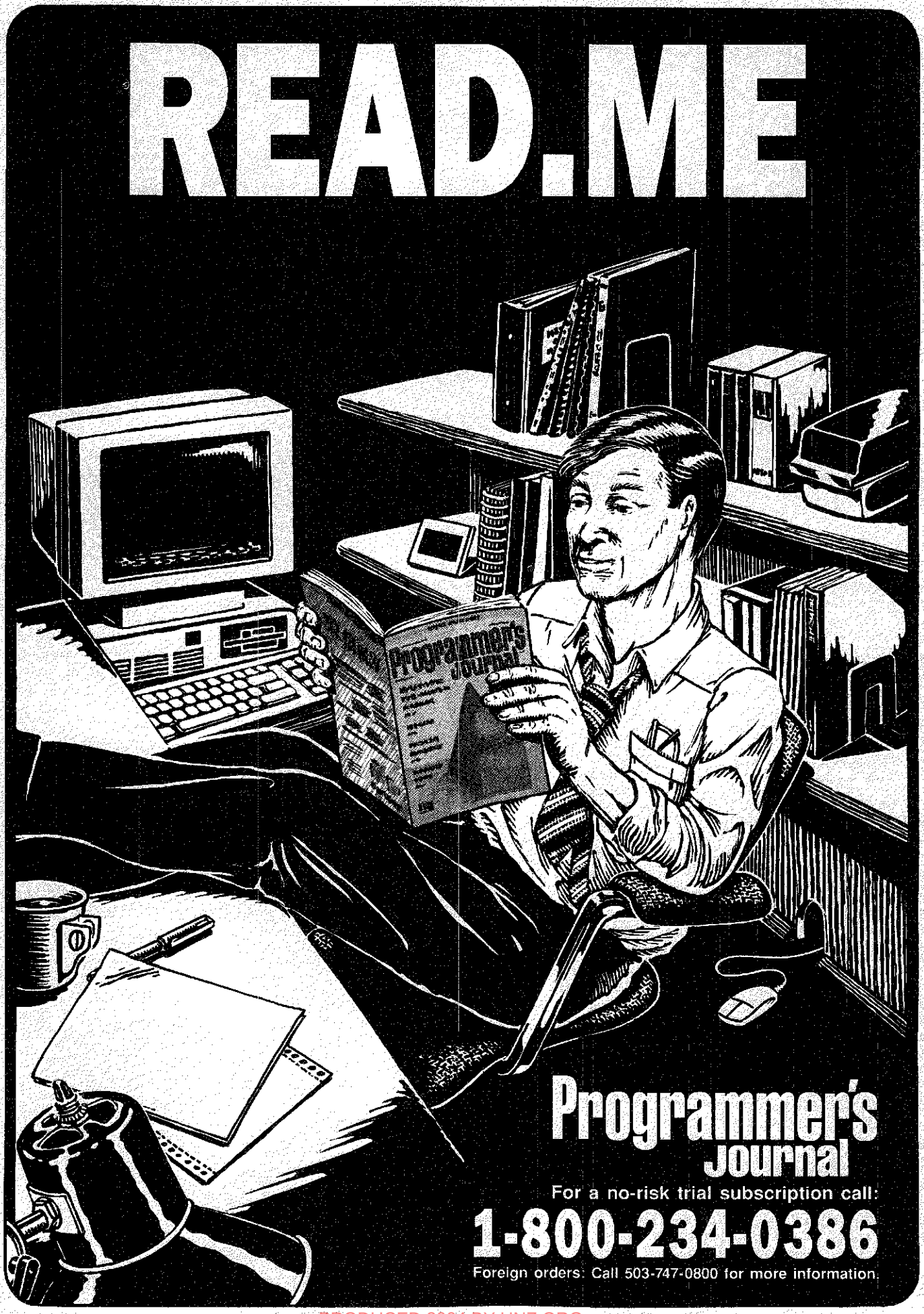
GREENLEAF Software©

Greenleaf Software, Inc.
16479 Dallas Parkway #570
Dallas, TX 75248

Circle 12 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

READ.ME



Programmer's Journal

For a no-risk trial subscription call:

1-800-234-0386

Foreign orders: Call 503-747-0800 for more information.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

environment and its management style. In addition, a well-thought-out professional development policy can be the fertilizer that stimulates and accelerates the growth process. Let's look at each of these areas in turn.

LET'S GET PHYSICAL

Even the most motivated software engineer cannot work efficiently in a poor work environment. Software engineering requires extreme concentration for hours at a time. Just organizing the problem in your mind before working on the solution can take 20 minutes or more—one interruption means the process must begin anew. All programmers, therefore, should have a place they can go to where they won't be disturbed, preferably a private office with a door. Small areas where two or three team members can work together are also helpful. DeMarco's and Lister's book *Peopleware* has some excellent ideas in this area.

TABLE 1: Pay me now or pay me later

Programmer Level	Cost/ Month	Months/ 1000 LOC*	Cost/ 1000 LOC	Average # of Errors	Error Cost/ 1000 LOC**	Total Cost/ 1000 LOC
Super	$8,000	0.5	$ 4,000	3	$ 750	$ 4,750
Average	$5,000	2.0	$10,000	15	$ 3,750	$13,750
Ineffective	$5,000	2.0	$10,000	80	$20,000	$30,000

*LOC = lines of code

**At a cost/error of $250

Take steps to minimize external interruptions, especially phone calls. A knowledgeable, friendly receptionist should screen project-team phone calls. A good electronic mail system also reduces work-related interruptions by allowing informal communication to be conducted at the receiver's convenience.

Each programmer should have a terminal that can be connected to any desired computer. A good back-support chair improves concentration during long hours in front of a terminal.

Each programmer should also have a set of manuals for the operating system, compiler, runtime library, and so on. Productivity is lost when the worker stops work, walks to a central library, tries to find the right manual (which is often misplaced), and then returns to work. The savings in time will pay for the manuals in short order. Store design documentation on-line, when possible, to improve accessibility and ensure that everyone is working from the latest version. A good software library is vital, as is an intelligent, motivated software librarian.

Tired of the CONFIG.SYS shuffle?

Let BOOTCON simplify your life.

"... it's bulletproof ... The interface is well thought out. You need this program."
Dr. Jerry Pournelle, *Byte*, July 1990

"Highly recommended."
John C. Dvorak, *PC Magazine*, July 1990

How many CONFIG.SYS and AUTOEXEC.BAT combinations do you have on your PC? BOOTCON eliminates the confusion by storing all your configuration information in just one set of configuration files. *Each time you boot your PC* you can select the configuration you want to use from a menu of up to 26 choices.

BOOTCON is ideal when you need to use conflicting programs such as Windows 3.0, QEMM/386, 386MAX, Soft-ICE, and Lotus 123. A default configuration will automatically be used if you do not make a selection within the specified time-out period. BOOTCON also provides an optional password protection feature.

BOOTCON lists for $59.95 and comes on both 5¼" and 3½" disks. It has an unconditional 30-day money-back guarantee. MS-DOS or PC-DOS 3.10 or later is required.

Modular Software Systems

115 West California Blvd., Suite 113
Pasadena, CA 91105

Phone (818) 440-9104 FAX (818) 440-9240

Windows and MS-DOS are trademarks of Microsoft Corporation. PC-DOS is a trademark of International Business Machines Corporation. QEMM/386 is a trademark of Quarterdeck Office Systems. 386MAX is a trademark of Qualitas Corporation. Soft-ICE is a trademark of Nu-Mega Technologies. Lotus and 123 are trademarks of Lotus Development Corporation.

Circle 13 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

IMPROVE MOTIVATION

Perhaps even more than the physical environment, your company's management style determines how successful you are at converting average programmers into super programmers. A good management style for programmer growth fosters motivation, focus, and planning. Following are some useful rules of thumb.

Promote healthy competition among team members. Encourage peer pressure (as opposed to pressure from above) and rewards.

Make it clear that if the project fails, every team member fails. Individual hard work is necessary, but it is not sufficient for advancement and official rewards.

Communicate project goals to all team members. Help them understand why the goals are important.

Lead but do not push. Give people as much responsibility as they can handle. Publicly express total faith in the ability of the individuals working for you. Trust your people.

If someone lets you down, express your disappointment in a frank discussion. Threats are not necessary or appropriate. Secure a commitment from the employee for improvement; then forgive as totally as if nothing happened. Find ways to publicly praise this employee.

If someone lets you down a second time, hammer them hard and fast. The atmosphere of trust and teamwork cannot work with employees who don't function in that environment.

Healthy conflict focuses on issues rather than on personal attacks. For example, a conflict about the best method of designing or coding a function is healthy; a conflict involving insults about a person's design or coding abilities is not. Avoid involvement in subordinates' healthy conflict—unless it is prolonged, in which case you should step in and resolve it. On the other hand, you should not tolerate *any* unhealthy conflict. Talk to the parties in private and put a complete and emphatic stop to the conflict.

THE VIEW FROM THE TOP

Professional development deals with long-range skill building. The first step is deciding which skills you will need two to four years in the future. The following categories may be useful: software development and related skills (such as new languages, database systems, and design approaches), hardware-related skills (network interfacing, I/O port control), computer science skills (artificial intelligence, parsing theory), and application-oriented skills (underwater acoustics, wave theory, accounting).

After deciding which skills will benefit your company the most, sit down with your staff for one-on-one informal discussions to determine their areas of interest. Ask that they take responsibility

We Have Ways To Improve Your Productivity

TE Developer's Kit

Why reinvent the wheel? TE Developer's Kit provides the most cost effective way of incorporating text editing features into your application. The kit includes the complete source code for the editor and the executable file. Supports Microsoft and Borland 'C' compilers.

Features: multiple files/windows, word-wrap and printing, edits files larger than available memory, reconfigurable keyboard commands and screen colors, undo, column block commands, and search/replace. **The kit also includes TER small editor routine. TER is a subset of TE. Your application calls this routine by specifying the window location and size, *maximum file size, and an input buffer or file. (DOS: $169, Windows: $269)***

WinMem: Windows 3 Memory Manager

WinMem provides an unlimited amount of virtual memory even in Windows' Real and Standard modes. **The device independence is not enough. WinMem offers you CPU and memory resource independence.** It provides an unlimited number of global memory handles, and reduces the overhead per block to 4 bytes (from 24 bytes of Windows). WinMem will trap any attempt to free an illegal memory block. This feature alone can save you a countless number of hours of debugging memory problems. Includes the complete source code. *($139)*

Spell Time *(NEW)*

Spell Time consists of a dictionary (over 100K words) and the routines to access the dictionary. Also includes an application specific dictionary, and a user dictionary. The routine will suggest alternates for a misspelled word. Highly optimized: 50 to 75 words/sec on a 12 Mhz 286 computer. Uses virtual memory on disk or EMS. An interface with TE included. Includes the complete source code. *($369)*

Sub Systems, Inc. 1-800-447-6819
Fax: 1-617-438-8901
159 Main Street, #8C, Stoneham, MA 02180, 1-(617)-438-8901

Circle 14 on Reader Service Card

ProGraphx Toolbox™

New Version 2.0 Write for a Free Demo Disk

NEW FEATURES INCLUDE:

- 320x200 EGA mode for games
- Icon Animation Editor
- Great for games, real-time,
- or detail-oriented graphics

Over 50 highspeed graphics routines for EGA/VGA 16 color modes, including proprietary color icon routines. **ProGraphx Image Editor**is a complete color icon paint package to edit and view up to 50 icons at once. Simply paint an icon, then use the **Prographx Toolbox™** routines to display it with amazing speed. Over 500 sample icons for use in your own programs. Includes **Operation: Airlift**, a full-screen scrolling arcade style game with source code. Works with ANY C or Turbo Pascal.
All routines written in assembly for fastest performance

SPECIAL OFFER
$99 with source
No Royalties!

To order, write:
Peder Jungck
1019 West Lovers Lane
Arlington, TX 76013
Incl. $5 S&H

Circle 15 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

for becoming proficient in some of these areas, stressing the professional opportunities that the new knowledge will open up. Present examples of how the knowledge will be useful. Convince them that becoming an expert in these areas will benefit them. Avoid having any staff member sign up for more than four of the areas.

The following suggestions can make it easy for each person to become an expert in their chosen area.

Find appropriate, reasonably priced evening classes (or short-duration day classes) taught in your area. Help your programmers register for these classes (one per semester); company funds pay for tuition and books. Find and order (at company expense) appropriate books. Allow your programmers to read the books at home; then place them in the company library. Or find books in your local university or public library. Check them out and lend them to the appropriate individual. If a book is lost, replace it with company funds if possible.

Subscribe to suitable technical journals and route them to the appropriate people (or better yet, take out a subscription for each person). Encourage programmers to take the journals home and read them. After everyone has read an issue, add it to your common library (or take out a library subscription).

Look for inexpensive personal computer software programs that involve the desired skills. For example, a low-cost ($49) Prolog compiler might pertain to AI expertise, a $99 Ada compiler might pertain to Ada knowledge, and so on. Buy appropriate software for your employees to use at home. Give a programmer a $49 Prolog compiler and manual, and I'll show you a new Prolog programmer trained for next to nothing.

Encourage your employees to use your computers after work for school and personal work. The added familiarity with the computer will greatly assist them in their work for you, and the added cost in electricity and wear and tear is insignificant.

Identify projects or project tasks that allow the employees to apply their growing skills. Encourage your staff to act as in-house experts in their designated areas. The process of researching and answering questions will educate and motivate them and will increase their confidence.

Perhaps my most important point is that you must *set a goal of developing your programmers into super programmers*. When the goal has been articulated, ideas abound for ways to accomplish it. I hope this article's suggestions are helpful, but the best ideas for your company are likely to be generated internally—often by the programmers themselves. DJ

Developer's Office Management Software

Turn Your Office Productivity Around!

Easy to use computerized system to organize, operate and control your software business. Written for DOS and DESQview API. ***The Developer's Assistant*** also works with Windows and Software Carousel.

Major features:	Other features	Affordable
User and Dealer indexed database		
"Bug" look-up DB for software support		
Superb invoicing capabilities		
Utilities i.e., serial # embedding program		
Directory of packagers, suppliers, etc.		
Directory for product releases		
Lots of reports and other features		
Supports both laser and dotmatrix printers		
Free 900 ☎ number to support your software		
Built-in calculator, phone dialer, other accessories		

Powerful • Integrated • Versatile • Slick

(512) 834-8927 $129.00 (+ tax in Texas)

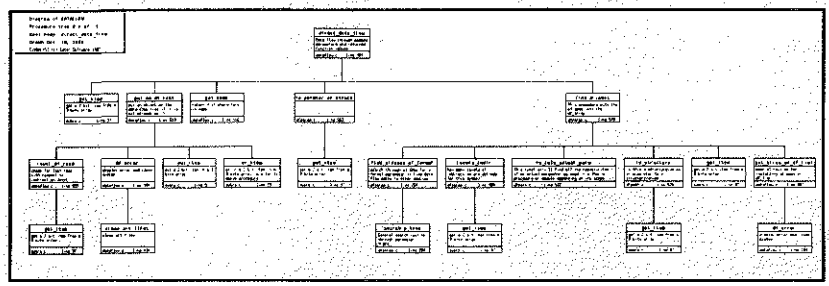
(VISA/MasterCard/AMEX accepted)

S.L.A.M. Software
3202 W. Anderson Ln. Ste. 208-176
Austin, TX 78757-1022

Circle 16 on Reader Service Card

C/ANALYST...

...THE HIGH POWERED DOCUMENTATION TOOL



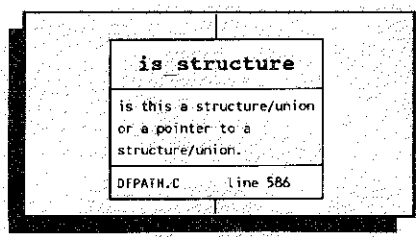
You have been handed a few inches of source code and have been asked to modify it. C/ANALYST can quickly bring you up to speed. It analyzes your C source and forms a database representing your entire program. From this database, C/ANALYST exhibits your program from different perspectives.

C/ANALYST is a set of four programs: ANALYZE, XREF, PROCTREE & DATAFLOW.

ANALYZE forms the database from your C source files. It has special features to process large multi-file programs quickly. Its built-in preprocessor saves include files in virtual memory as a stream of tokens to eliminate rescanning and redundant accesses to disk. Large hash tables provide quick access when there are thousands of symbols present in the code. Using virtual memory, ANALYZE can build databases much larger than the available memory.

XREF generates a function index followed by a global cross reference listing. The function index alphabetically lists each function, gives its location and its definition taken from comments in the source code. The global cross reference not only shows where each object is used but how it is used as well. Read and Write codes along with indirection levels are included with each reference. Objects are referenced to the function which uses them. Structure members are listed individually.

PROCTREE generates procedure tree diagrams which show the overall control flow of the program. Diagrams can be printed on HP LaserJet printers or sideways on dot matrix printers. Although the diagrams can be almost unlimited in size, a heuristic pruning method chops trees into multiple diagrams of reasonable size.



DATAFLOW searches through the database to find the flow of information between each module pair in the program. It shows flow through global variables when one module writes to a variable and another module reads that variable. Parameter passage to functions can be shown along with symbolic function returns. A sophisticated technique searches for indirect data passage through formal parameters which are pointers.

FOR MORE INFORMATION CALL US TODAY

(503) 581-5622

VERSION 2.0 $250.00 ppd.
+ 15.00 for overseas

30 day money back guarantee
requires MS-DOS 2.0 or above.

CATER SOFT WARE

2182 Westfarthing Way NW
Salem, OR 97304

Circle 57 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Announcing version 2:

Victor Image Processing Library

Use Victor to develop powerful image applications

Work with images of any size -- use conventional, expanded, and extended memory

Now your applications can support 8-bit color and gray scale images of any size because Victor gives you complete control over conventional, expanded, and extended memory.

Display on Super VGA

Display images on EGA/VGA and super VGA up to 1024 x 768 256 colors.

Load & save PCX/TIFF/GIF/BIN

Handle images from any source, or create translation programs between the popular file formats.

Gray scale and color images

Powerful image processing for all images -- your software can have features like: zoom, resize, brighten, contrast, sharpen, outline, linearize, matrix conv, colorize, & more.

ScanJet and LaserJet support

You can offer device control for gray scale scanning -- AND print halftones at any size.

Victor supports Microsoft C, QuickC, and TurboC, includes demonstration and prototyping software, and full documentation. Source code available.

Image-based applications can be developed in MSC, QuickC, and Turbo C environments.

Your application will be able to print halftones at any size.

Give your applications support for scanner and laser printer.

***Victor Library version 2*, $195**

Call (314) 962-7833 to order VISA/MC/COD

Catenary Systems 470 Belleview St Louis MO 63119 (314) 962-7833

Circle 17 on Reader Service Card

Passive Backplane CPUs

PC Tech manufactures a full line of modular passive backplane high performance PC compatible CPUs.

Passive backplane based systems have many advantages, including ease of assembly, flexibility, fast service (by swapping boards), and simple upgrades to new or faster processors.

Contact us for more information, pricing, and benchmark results.

80386SX:	20 MHz, 512K through 16M DRAM
80386DX:	25 MHz & 33 MHz, 1M - 32M DRAM, 128K cache module optional
80486:	25 MHz & 33 MHz, 1M - 32M DRAM, 256K cache module optional

Call, write, or FAX for complete specifications and retail or OEM terms. All PC Tech products are available in complete systems as well as components for retail, OEM, and private label sales.

Designed, Manufactured, Sold and Serviced by:

PCTech Inc.

Voice . . . (612) 345-4555
FAX . . . . (612) 345-5514
Modem . . (612) 345-4656

907 N. 6th St., Lake City, MN 55041

Circle 18 on Reader Service Card

Free Yourself!

With Screen Manager Professional™

Concentrate on the substance of your programs with the advanced interface design library for C programmers.

Screen Manager Professional gives your applications the finest bullet proof windowing interface available. All the functions have been carefully designed and tested with the serious programmer in mind The functions are easy to use and remember. The elegant and powerful functions provide the fastest possible response and smallest possible size. Free yourself to concentrate on the substance of your program. Over 200 functions provide all the windowing tools needed to develop serious applications whether you are a professional or a hobbyist.

SCREEN MANAGER Professional™

MAGEE™

Call today for a free demonstration diskette
1-800-662-4330

P.O. Box 1587
Norcross, GA 30091 USA
404-446-6611
BBS 404-446-6650

SMP gives your applications:

- Windowing
- Menus
- Mouse Support
- Field oriented data entry
- Interactive field building utility
- Keyboard Support Functions
- Context Sensitive Help with Hypertext
- Low RAM Overhead / High Speed
- User attachable validation functions

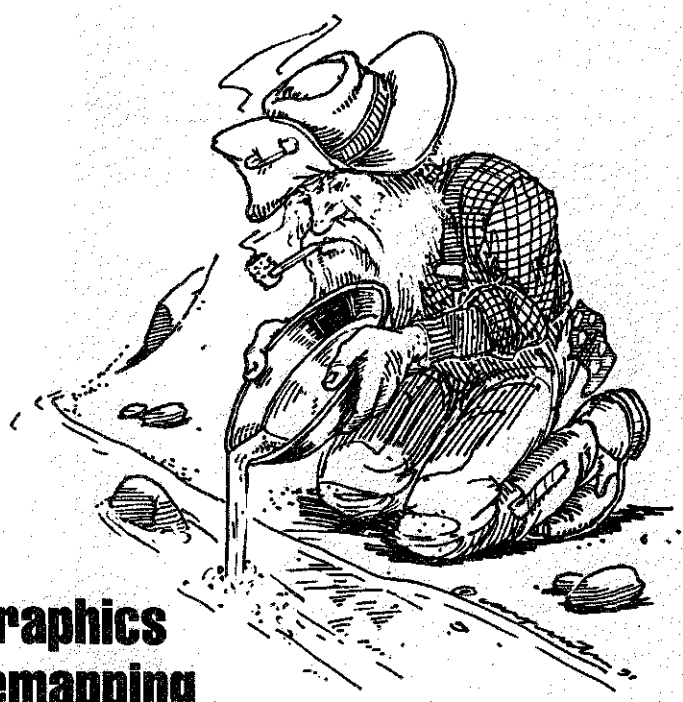
Linkable with: Microsoft C - Zortech C, C++ - Quick C
Turbo C, C++ - Watcom C

Only $349.95 **Source code available**

Circle 19 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Pixel panning



Getting the most out of your graphics adapter with video memory remapping

By Peder Jungck

Graphics resolution and color are attractive features of the EGA and VGA, but some of their other useful features are often overlooked. The ability to pan the screen image both horizontally and vertically can greatly expand EGA/VGA's usefulness. This article explains how to remap the video memory to allow this kind of pixel panning.

To understand the value of panning, consider video arcade games. Their appeal is their full-color display of immense playing fields and the apparent "movement" of the animated characters, vehicles, and so on. Video game designers are able to achieve these effects not only because they work with sophisticated graphics hardware, but also because they have a thorough knowledge of that hardware.

The key to moving relatively complex graphics around quickly lies in letting the video board move them rather than moving them in software. This is accomplished by changing the memory offset where the graphics begin to display on the screen.

If you study video games that have full-screen panning or scrolling—such as Nintendo or arcade games—you'll notice that the background stays fixed. Game designers create a detailed background that remains fairly unchanged, while objects move around in the foreground. This trick allows fast operating speeds, because the background exists in the hardware and requires virtually no software time. The few foreground objects can be dealt with simply and quickly.

NOW FOR THE BIG PICTURE

Expanding your use of EGA/VGA to allow panning and dynamic video memory configurations can be accomplished by following the procedures described below.

When you set a video mode through the video BIOS interrupt, all characteristics are usually set to the defaults. In video mode 10h (EGA 640 × 350 × 16 colors), the memory is mapped to 80 bytes per scan line. The 81st byte is the first byte of the second scan line (see Figure 1). Because there is no offscreen data, you can't pan horizontally. By remapping the video memory, you can tell the video adapter that there are 160 bytes per line and have the second scan line start with the 161st byte. The display will still show only 80 bytes' worth of data at a time, but you will be able to move the data horizontally.

You can help yourself visualize this by placing a diskette on a sheet of paper. The paper represents the video memory, and the portion underneath the disk is what you will see on the display. The video memory is wider and taller than the disk, thus allowing four-way panning.

Video mode 10h is not a very good choice for extensive panning on the EGA adapter because it has only a little over two pages' worth of memory. Although an EGA adapter is thus limited to 256K of memory, a standard VGA (640 × 480) can handle even fewer pages—about one and a half. But mode 0Dh (320 × 200 × 16), used in many recent video games, has enough memory for eight pages (see Figure 2). With this much memory to play with, it's easy to have multiple arrangements for panning. You could pan vertically or horizontally for eight pages, for example, or you could pan two pages horizontally and four vertically. I will illustrate panning with mode 0Dh throughout this article, but these techniques can be applied to any EGA/VGA video mode.

NONSTANDARD MEMORY LANE

To set up and use the video board with a nonstandard memory map for panning, follow the procedure outlined below. First, initialize the video mode through the standard BIOS routine. Then set the bytes-per-line register on the video board as well as within your software. Make sure that all line, circle, bit-block, or other

Peder Jungck is a software engineer in Dallas, TX. He is the author of the ProGraphx Toolbox EGA/VGA graphics library. Readers may contact Peder by writing to him at 1019 West Lovers Lane, Arlington, TX 76013.

ILLUSTRATION BY JIM CARPENTER

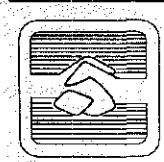
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LISTING 1

```
; PIXELPAN.ASM -- EGA/VGA Pixel Panning routines
; BY PEDER JUNGCK ( Copyright 1991 )
; Assembly language routines designed for C-style calling conventions
; All example routines use a near call model - assemble with /MX switch
; NOTE: the SetLineLength function is error trapped for Mode 0Dh (320x200)

SLOW_EGA  equ  0
        .model small

        .code
Public  _Video_Mode             ; (int Mode)
Public  _SetLineLength          ; (int Length)
Public  _SetViewPosition        ; (HPixel,LineNum)
Public  _Set_Pixel              ; (x,y,Color)

BytesPerLine    dw      40      ; screen width in bytes (320x200 Mode)
VOffset         equ     0a000h  ; video RAM offset

Video_ModeParms struc
                dw      ?       ; pushed BP
                dw      ?       ; return address pushed by call
    vm_mode     dw      ?       ; pass new video mode parameter
Video_ModeParms ends
_Video_Mode     proc    near    ; void Video_Mode(int Mode);
        push    bp
        mov     bp,sp           ; put sp in bp, to get parameters on stack
        mov     ax,word ptr [bp+vm_mode]
        xor     ah,ah           ; set video mode ( ah=0 )
        int     10h             ; call video BIOS
        pop     bp
        ret
_Video_Mode endp

SetLineLengthParms   struc
                dw      ?       ; pushed BP
                dw      ?       ; return address pushed by call
    LineLen     dw      ?       ; pass new scan line length
SetLineLengthParms ends
_SetLineLength  proc    near    ; int SetLineLength(int Length)
        push    bp
        mov     bp,sp           ; put sp in bp, to find parameters on stack

        mov     ax,word ptr [bp+LineLen]
        cmp     ax,40           ; Minimum Line Length (320x200) 40 bytes
        jae     Low_OK          ;  1 page wide x 8 high (mode 0Dh default)

        mov     ax,40           ; Must be at least the minimum length
        jmp     Hi_OK

Low_OK: cmp     ax,320          ; 320 bytes wide leaves only 200 high
        jbe     Hi_OK           ;  8 pages wide and one high (mode 0Dh)
        mov     ax,320          ; Must be less or equal to maximum

Hi_OK:  mov  BytesPerLine,ax
        xchg    bx,ax           ; save parameter to BX
        mov     dx,3d4h
        mov     ax,13h
        out     dx,ax           ; BytesPerVideoRamRow Func
        inc     dx              ; 3d5h data port
        mov     ax,bx           ; number of lines to output
        shr     ax,1            ; need words for port
        out     dx,ax

        mov     ax,bx           ; return # of lines we set
        pop     bp
        ret
_SetLineLength  endp

SetViewPositionParms   struc
                dw      ?       ; pushed BP
                dw      ?       ; return address pushed by call
```

continued

Turbo Pascal/Turbo C Users:



Come join us at . . .

GeTUGether '91!

Spend a weekend with internationally known experts in Turbo Pascal, C++ and Assembler programming. You'll enjoy 2-1/2 days of small group sessions, like Terry Hughes (TurboPower) on objects for async communications; Michael Burton (KeyTronic) on designing user-customizable interfaces; and Jeffrey Watson (GALI) on pointer math. You'll also hear other speakers like Jeff Duntemann (PC Techniques); Bill Gates (Midnight Engineering); Richard Tom (TEGL) and many others, including Borland staff people.

GeTUGether '91 will be held July 19-21 in Silverdale, Washington (near Seattle). Conference fees are very affordable ($145 for individuals, which includes 3 meals during the conference). A corporate option, which includes all materials and videos made at the conference, is also available. Call us for the latest information – and we'll see you here!!

Turbo User Group

Post Office Box 1510 • Poulsbo, Washington 98370

(206) 779-9508

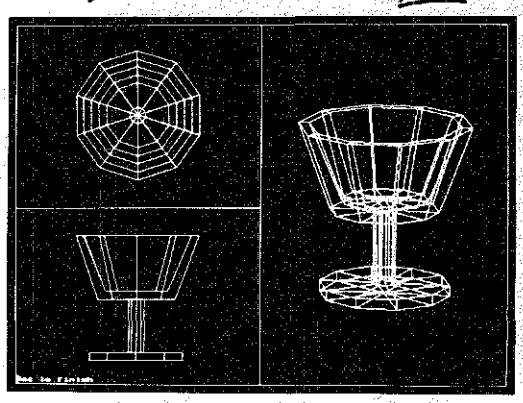
Annual dues (US funds): U.S. $27.00; Canada $32.00; Overseas $39.00

Circle **20** on Reader Service Card

THE ULTIMATE CAD/CAM ENGINE

TurboGeometry Library 3.0!

Helps You build it right THE FIRST TIME!



Over 300 2D & 3D device independent geometric routines. Surfacing; Spherical & Solid Geometry; Tangents; Curves; Hidden Line; Polygons; Perspectives; 2D & 3D Clipping; Link List Library & more. In C or Pascal, w/source. 400 pg manual. $199.95, Foreign $225.00. Prices incl S&H. 30 day guarantee. VISA, MC, PO, Chk.

2116 E. Arapaho Rd.
Suite 487
Richardson, TX 75081
USA

Disk Software, Inc.

FAX or Phone
214-423-7288
Toll Free
1-800-635-7760

Circle **21** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

SLATE

Universal Printer Driver

NEW! Text & Graphics!

By including SLATE, you can print both **Text and Graphics** on **over 400 printers. Immediately! Painlessly!**

You can use SLATE in your product with **no royalties.**

Make your product more functional and competitive by taking advantage of SLATE's **advanced text features:**

- Support multiple printers on the same system.
- Output to parallel printers, serial printers, DOS files, console, DOS print spooler, and Novell network printers.
- Support proportional fonts laser printer soft fonts.
- Set exact print positions.

You can add the **NEW** Graphics Subsystem for **advanced graphics features:**

- Print images from the screen, graphic files, or custom image systems.
- Scale and Rotate the printed image.
- Convert colors to patterns.
- Print grey scale (shaded) and color images.
- Intermix text and graphics.

What is SLATE?

SLATE is a set of C libraries with over 150 text printing functions, a Database of over 400 printers, and tools for end user configuration and testing.

SLATE with Graphics adds over 60 graphic printing functions.

Call now for more information. Order **SLATE for $299 or SLATE with Graphics for $448** with our risk free, 30 day return policy.

We accept Visa, Mastercard, COD's or PO's from qualified companies. Source code, maintenance, and site licenses are available. Also ask for information about our S_PRINT Text Formatting System for Software Developers.

800-346-3938

The Symmetry Group

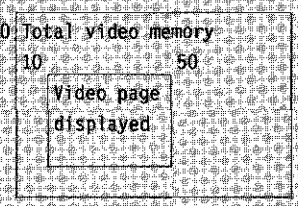
PO Box 26195
Columbus, OH 43226
614-431-2667
FAX 614-431-5734

Circle 22 on Reader Service Card

FIGURE 1: Memory map and scan lines, mode 10h

1	0	1	2	.	.	.	.	78	79
2	80	81	82	.	.	.	.	158	159
3	160	161	162	.	.	.	.	238	239
4	.	.	.	.	.	.	.	.	.
5	.	.	.	.	.	.	.	.	.

FIGURE 2: Memory organization, mode 00h



In mode 0Dh a page is only 40 bytes wide (320 pixels).

routines adapt to the new bytes-per-line setting. Next, set your initial view position within the display memory.

Now you can draw any image anywhere within the display memory. Make sure that the graphics routines you use to display images adjust for the new bytes-per-line memory map. When all images and background have been generated, you can call the **SetViewPosition** routine (see Listing 1) to put your viewport into the display memory.

Listing 2 illustrates some set-up code that provides a simple example of pixel panning. The first step calls the **Video_Mode** function with 0Dh (320 × 200 × 16) as the parameter.

The second step sets the line length to 80 bytes (normally 40 bytes). The EGA/VGA hardware actually uses a number of two-byte words to denote the parameter for the width of the video memory. For example, the 80 we passed as a line-length parameter gets stored in the video hardware as 40 words.

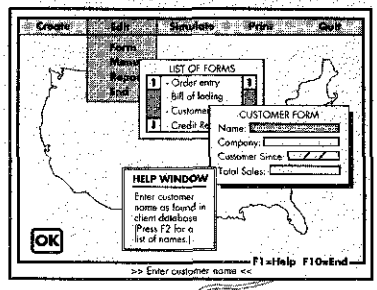
COORDINATE CAREFULLY

For the third step, **SetViewPosition** takes the pixel coordinates of the upper left corner of the region you wish to display as the *x* and *y* coordinates. In mode 0Dh, the valid range for *x* in our example would be from 0 to 320. The first page would be from 0 to 319, and the second page of

POWERFUL SCREENS FOR POWERFUL PROGRAMS

HI-SCREEN XL is a powerful screen management system with a high-level modular approach that provides for superior programming **control, flexibility and speed.**

- The built-in **screen editor** is used to create screens/windows/menus along with all field descriptions. Any **Hercules, CGA, EGA** or **VGA** graphics can also be captured. Screens are then saved in a compressed file format outside your program.
- **High-level calls** from your program give you total control over the user interface. Combine text and graphics, dialog boxes, context-sensitive help, elaborate data entry & menu management, keyboard & mouse control, etc.



- Supports all versions of **C, Basic Cobol, Fortran, Pascal, dBase, Assembly,** etc.
- Use linkable (with C and QuickBasic) or removable TSR screen driver (30-40K).
- 2 compatible versions:

HI-SCREEN XL.....$149
HI-SCREEN XL *Professional Series*
(with mouse & graphics support):.....$325

30 day money-back guarantee • No royalties

Call today: **1-800-338-2852**
in CA: 415-896-0708 • Fax: 415-896-0709

SOFTWAY, Inc.
185 Berry Street, Suite 5411,
San Francisco, CA 94107

CALL FOR DEMO

Circle 23 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

memory off to the right would be from 320 to 639. If you set the value to be greater than 320, the rightmost portion of the display would wrap around to use the leftmost portion of video memory for the next scan line. Note that there are no fixed bounds laid out in the video board's memory. Only those rules you set up are going to be followed, so be careful.

Because mode 0Dh has eight total pages of video memory, using two pages horizontally leaves you with four vertical pages. This allows a range for *y* of 0 to 600. Pages 1 and 2 are from 0 to 199, Pages 3 and 4 are from 200 to 399, 5 and 6 are from 400 to 599, and 7 and 8 are from 600 to 799.

If you use numbers greater than 600 for the *y* coordinate, the offset address that is passed to reference the video RAM when refreshing the display can have strange effects. Note that a small portion of leftover memory remains at the end of the 64K video memory buffer. Also, addresses may wrap around in the 64K page. Just remember that the video RAM is laid out as a single 64K page of DOS memory and that latches determine which of the video planes you access. Step three simply sets the offset to the upper left corner.

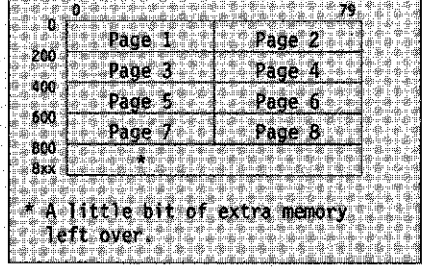
READY, SET, VIEW!

Now you can add some graphic objects to the screen that you wish to pan around. The **set_pixel** routine is a sample graphics primitive that adapts to the **BytesPerLine** value. Many of Michael Abrash's line and circle drawing examples make use of a variable called **BytesPerLine**. By modifying your code to adjust for the number of **BytesPerLine** (80), you can get images to display properly.

If you use unmodified code, the first line of an image will display in Page 1, and the second line of the image will display on the same scan line, but in Page 2 (see Figure 3). The image's third line will display just below its first line on Page 1, and so on. By properly setting the **BytesPerLine** in the code, you tell the routine to skip those extra 40 bytes that make up the rest of the scan line in Page 2.

In Listing 2, I use a couple of loops to call **SetViewPosition** and pan the display—first across and then down and to the left. Notice that all coordinates passed are actual *x, y* positions and do not deal with an offset within a page. To display Page 4 you would call **SetViewPosition** (320, 200) rather than calling a page setting function of some sort.

FIGURE 3: Memory map of mode 0Dh with 80 BytesPerLine set



FOREWARNED IS WISER

The one disadvantage of pixel panning is the lack of standards about when EGA and VGA cards actually remap the video. For some older EGA and VGA cards, you'll need to set the SLOW_EGA define in Listing 1 to add another wait loop. Newer VGA cards don't seem to need this. In spite of this drawback, panning and rearranging your video memory map can be a powerful and effective tool. Implementing these procedures points out the importance of having adaptable graphics routines at your disposal. PJ

```
    Hpixel      dw      ?           ; pass in horizontal pixel offset
    LineNum     dw      ?           ; pass in line number
SetViewPositionParms    ends
_SetViewPosition proc near    ;void SetViewPosition(int HPixel,int Linenum)
        push    bp
        mov     bp,sp               ; put sp in bp, to find parameters on stack
        mov     ax,[bp+HPixel]      ; get horizontal pixel address
        mov     cx,ax               ;  and save in cx
        and     cx,07               ; get pixel offset of position
        shr     ax,1
        shr     ax,1
        shr     ax,1
        xchg    bx,ax               ; and save in bx
        mov     ax,[bp+LineNum]     ; get number of lines
        mul     BytesPerLine        ; multiply times bytes per line
        add     bx,ax               ; new upper left corner offset
        mov     dx,3dah
wait:   in      al,dx               ; make sure not already doing a retrace
        test    al,08h
        jz      wait
Retrace: in     al,dx               ; wait until the start of retrace
        test    al,08               ;  to perform clear of 3c0 port
        jnz     Retrace
        cli                         ; disable interrupts
        in      al,dx
        mov     ax,bx               ; upper corner in AH, BH holds high value
        mov     al,0ch              ; hi byte offset
        mov     dx,3d4h
        out     dx,ax
        mov     al,0dh              ; low byte offset
        mov     ah,bl               ; bl holds low byte value
        out     dx,ax
if SLOW_EGA
        sti
        mov     dx,3dah
wait2:  in      al,dx               ; wait til we start vertical retrace
        test    al,08h
        jz      wait2

        cli
endif
        mov     dx,3c0h             ; ATC index register
        mov     al,33h              ; palette address source 1, reg address 13
        out     dx,al
        xchg    cl,al               ; get pixel offset
        out     dx,al

        sti                         ; enable interrupts
        pop     bp
        ret
_SetViewPosition        endp

Set_PixelParms          struc
                dw      ?           ; pushed BP
                dw      ?           ; return address pushed by call
    ps_xpos     dw      ?
    ps_ypos     dw      ?
    Color       dw      ?
Set_PixelParms          ends
_Set_Pixel proc near                ; void Set_Pixel(int x, int y, int Color)
        push    bp
        mov     bp,sp
        push    ds                  ; save the data segment
        mov     ax,VOffset          ; get the video RAM address
        mov     ds,ax               ; set data segment to EGA/VGA
        mov     ax,[bp+ps_ypos]     ; get the y coordinate
        mul     BytesPerLine        ; get the offset to line y
        mov     bx,[bp+ps_xpos]     ; get the x coordinate
        mov     cx,bx               ;  and save in CX too
        shr     bx,1                ;  and divide by 8
        shr     bx,1
        shr     bx,1
        add     bx,ax               ; bx := BytesPerLine*y+(x/8)
                                    ; we now have x,y memory offset
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
and     cl,7            ; generate mask for one pixel in byte
xor     cl,7
mov     ch,1
shl     ch,cl           ; ch := 1 >> (7-(x mod 8))
mov     dx,3ceh         ; now select write mode 2
mov     al,5
out     dx,al           ; select mode (register 5)
mov     dx,3cfh
mov     al,2
out     dx,al           ; and set to write mode 2
mov     dx,3ceh         ; set up mask
mov     al,8
out     dx,al           ; select mask (register 8)
mov     dx,3cfh
mov     al,ch
out     dx,al           ; set mask
mov     al,[bx]         ; read from card and color dot
mov     ax,[bp+Color]
mov     [bx],al         ; write color to dot
mov     dx,3ceh         ; restore default mode
mov     al,5
out     dx,al           ; select mode (register 5)
mov     dx,3cfh
mov     al,0
out     dx,al           ; set write mode back to 0
mov     dx,3ceh
mov     al,8
out     dx,al           ; select mask (register 8)
mov     dx,3cfh
mov     al,0ffh
out     dx,al           ; and set default mask
pop     ds              ; restore data segment
pop     bp
        ret
_Set_Pixel      endp
end
```

LISTING 2

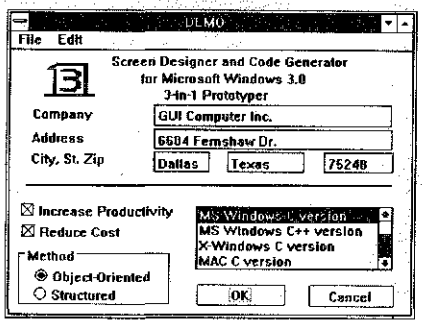
```
/* PAN-TEST.C --- Display Color Blocks and then pans the display      */
extern void Video_Mode (int Mode);      /* Use BIOS to set mode       */
extern int SetLineLength (int Length);  /* Sets bytes/scan line       */
extern void SetViewPosition(int HPixel, int LineNum); /*Upper left
                                                pixel to display on screen  */
extern void Set_Pixel(int x, int y, int Color); /* Set pixel to color  */
int main () {
    int i,j,k,l;                /* some loop variables                */
    Video_Mode ( 0x0d );        /* Step 1: Set video mode to 320x200x16   */
    SetLineLength( 80 );        /* Step 2: 80 Bytes wide virtual screen   */
                                /* Sets up 2 wide by 4 high memory    */
    SetViewPosition( 0,0 ); /* Step 3: Display upper left page        */
 /* Step 4: Place graphics in video memory -- Make sure whatever
  * routines you use properly adjust for the new memory layout.       */
    for (l=0; l<20; l++)            /* Draw a color checkerboard      */
        for (k=0; k<20; k++)        /* Not all is initially visible   */
            for (j=0; j<18; j++)    /* but will be revealed by panning */
                for (i=0; i<32; i++)
                    Set_Pixel( i+j*32, k + l*20, j + l);
/* Step 5: now we pan a little */
    for (i=0; i<320; i++)           /* 320 is first position page 2   */
        SetViewPosition( i,0 );     /* Do a Horizontal Pan first      */
    for (i=320; i>0; i--)           /* Diagonal Pan down and to left  */
        SetViewPosition( i,320-i ); /*  to page 3/5 (120 lines down)  */
    for (i=0; i<320; i++)           /* Vertical Pan up to origin      */
        SetViewPosition( 0,320-i ); /*  to page 1 (120 lines down)    */
    Video_Mode ( 3 );               /* Restore to text mode and exit  */
};
```


C/C++ Windows 3.0

Screen Designer Code Generator

Only $99

The easiest way to learn and to master Microsoft Windows programming



With 3-in-1, developing this window screen requires only

20 Minutes !!

That includes screen layout, code generation, compiling and linking.

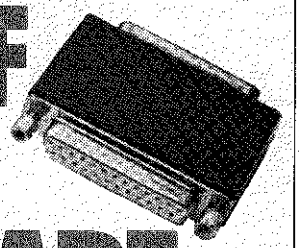
GUI's 3-in-1 C $99 C++ $159 Both $199

30 Day Guarantee · No Source Code Royalties

Also many low cost Windows source code utilities.

1-800-800-9010 TEL 214-250-3472 Fax 214-250-1355

DON'T LOSE 1/2 OF YOUR SOFTWARE PROFITS!



Save your lost profits with SecuriKey — a unique hardware device that prevents unauthorized use and duplication of computer software.

- IBM, compatible & Macintosh
- Allows unlimited backup copies
- Does not permit duplication
- Can record software usage
- Available in custom colors & logos
- Variety of programmable options
- UNIX/XENIX

FOR FURTHER INFORMATION, CALL TODAY!

1(800) 456-2587

micro security systems, inc.

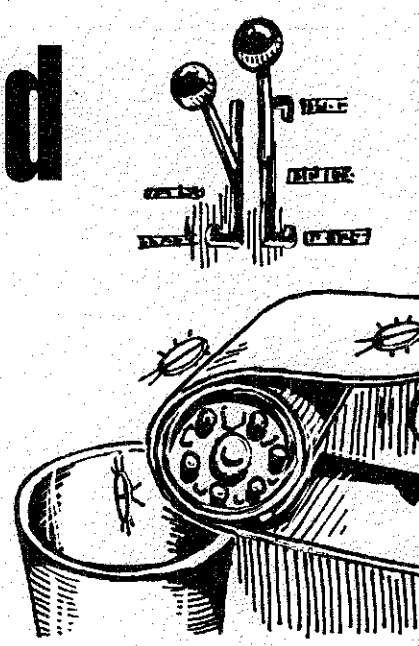
4750 Wiley Post Way, Suite 180
Salt Lake City, Utah 84116
FAX (801) 575-6621

Contact this advertiser directly.

Circle 47 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Hardware-assisted debuggers



Powerful hardware tools for complicated problem solving

By Brett Salter

The software-only debuggers that come with assemblers and compilers are usually designed for solving problems at the applications level. Although some of these products can also help debug terminate-and-stay-resident (TSR) programs, it is difficult or impossible to use them when debugging problems involving device drivers, hardware interrupt service routines, real-time interactions between two or more programs, or a hung system. Hardware-assisted debuggers use a combination of hardware and software to enable you to solve these more complex kinds of problems. Levels of hardware assistance in these debuggers include a break-out switch, protected or hidden memory, real-time breakpoints, and a real-time trace buffer.

The push-button **break-out switch** is used to generate a nonmaskable interrupt (NMI) to recover from a hung system and activate the debugger. If your program disables hardware interrupts via a CLI instruction, then all hardware interrupts—including the system timer, keyboard, mouse, and communications interrupts—are disabled. The NMI's great advantage is that it can still get through.

Luckily, IBM designed the ISA bus so that the NMI signal is available for use by add-in cards to signal a parity error. Most debuggers support the use of NMI to wake them up from a hung system (although their ability to function after being awakened varies by product). It is easy to generate an NMI from the bus: hook a momentary-contact switch between pins A1 (I/O channel check) and B1 (ground) of the ISA bus. There's no need for any debouncing circuitry, because the NMI signal goes through circuitry that effectively prevents multiple NMI signals from reaching the CPU.

SMALL FOOTPRINTS ARE BEST

Protected memory is RAM that contains the debugger code and data. It can be read and written while the debugger is active, but when the debugger is inactive it cannot be written. In general, RAM is preferable to ROM, because software updates can be distributed via a diskette instead of via ROM chips. The protected memory keeps the debugger from being corrupted by a runaway program. Usually, this memory is also "hidden" from the program under test. On a PC, the classic way of hiding this memory is to use some memory in the region between 640K and 1MB. As long as the memory address is flexible and the amount of memory address space required (the footprint) does not preclude the use of desired add-in cards, this solution is effective for systems based on CPUs from the 8088 to the 80486 and beyond.

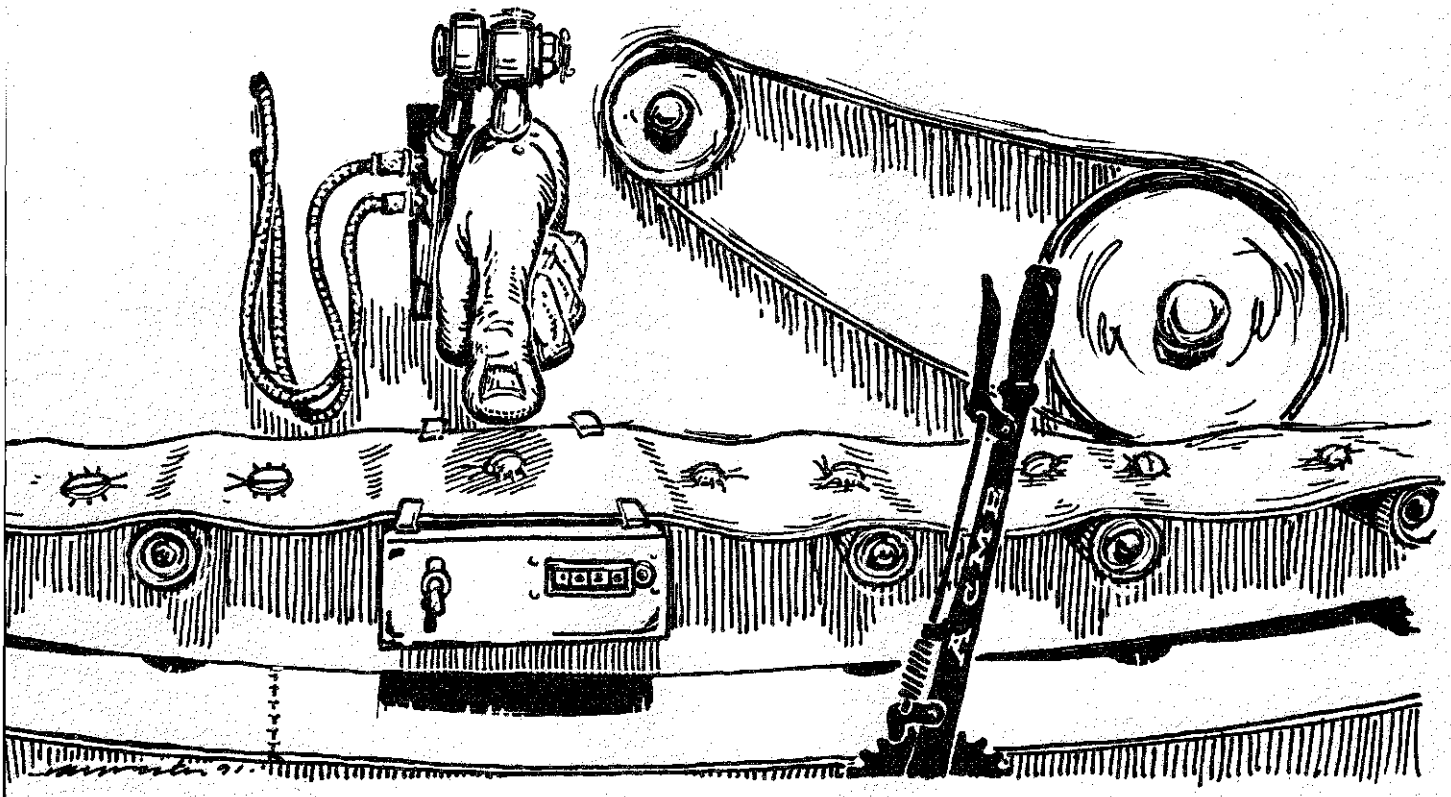
The debugger's need for 100K or more of memory and the desire to keep the total footprint small can both be met using a paged memory scheme like that used by expanded memory boards. The memory on the Periscope Model I board is structured this way. The board has 512K of memory with four 8K pages, consuming a total footprint of 32K.

Because the debugger is resident in its own write-protected memory, a very interesting capability arises. If the board is addressed in the region of memory that is scanned by the BIOS for add-in ROM, you can boot the system normally and install the debugger software so that, when the system is rebooted, the debugger's memory will identify it as add-in ROM. In this way the debugger gains control very early in the boot process, allowing you to step through other add-in ROM and the remainder of the BIOS start-up code.

On 80386 or later CPUs it is possible to have the debugger execute in protected mode and run your application in virtual 8086 mode. This scheme keeps the debugger out of the application's address space and protects the debugger from the application. The drawback of this method is that it precludes the use of another 386 control program, such as

Brett Salter is the president and founder of the Periscope Company, a firm that specializes in hardware and software debugging tools for the PC market. Brett can be contacted at (404) 875-8080.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



386MAX or QEMM, because only one program can be controlling the processor's protected mode.

A more flexible solution is to use the memory manager to provide the protected memory. Periscope and the 386MAX manufacturer, Qualitas, have done just that. 386MAX Version 5.11 remaps extended memory to emulate the paged memory provided by the Periscope Model I board. This makes most of the board's functions, including write-protected and hidden memory, available without any special-purpose hardware.

EFFECTIVE HARDWARE

Real-time breakpoints allow a hardware device to watch the bus or CPU for an event, such as a memory write, and to activate the debugger when the desired breakpoint occurs. To be effective in a variety of situations, the hardware must support different platforms, must have no adverse effects on the system under test, and must not change the characteristics of the program being debugged. For example, when debugging a communications program, you don't want to add wait states to your system or have a software-based debugger cause it to run more slowly; either situation could cause the problem you are chasing to change or disappear.

Starting with the 80386, CPUs contain built-in debug registers that support limited real-time breakpoints on instruction execution, memory write, or memory read/write. A maximum of four breakpoints are included; each can cover a range of a byte, a word, or a double word. These breakpoints can be useful if you know the exact memory location, but they're useless for memory ranges larger than four bytes. To do its job, a board that supports real-time breakpoints must watch the following information as it occurs:

- **Address signals:** on the 80286 and 80386SX CPUs, the address signals use 24 bits. On the 80386 and 80486, they use 32 bits.
- **Data signals:** on the 80286 and 80386SX, the data signals are 16 bits wide. On the 80386 and 80486 they are 32 bits wide.
- **Status information:** this information indicates what the CPU is doing. The possibilities include memory read, memory write, instruction prefetch, I/O port read, I/O port write, and less common states such as interrupt acknowledge, halt, and shutdown.
- **Byte-enable signals:** these bits indicate which of the two or four bytes of data is valid for the current event. If the program is writing one byte to memory at 9000:0, only the low byte is valid—any data in the other three bytes must be ignored.

For example, assume that you're debugging a communications program and need to stop when your program reads I/O port 3FEh and bit 4 is high. The following items must all be checked in real time:

- The address must be 03FEh.
- The status information must indicate an I/O read.
- The byte-enable signals are used to select the valid byte (byte 2 in this case, assuming byte 0 is low and byte 3 is high).
- Bit 4 of byte 2 must be set.

FASTER THAN A SPEEDING BUS

Early real-time products for the PC family were able to monitor the activity of the CPU from the system bus. This approach worked well on the PC, XT, and true-blue ATs, because all of the useful signals were echoed out to the bus.

If you're running a CPU that is 10-MHz or faster, it doesn't make sense to slow the CPU down to 8 MHz so that it can echo everything out to the system bus. Manufacturers, therefore, moved to private buses for the things that needed to be faster than 8 MHz, such as main memory. Support was retained for the memory and I/O ports used on add-in cards, but because most of the "action" occurs in main memory, the information-gathering had to switch to the CPU. The only item that was lost in the move was the access to DMA signals. DMA is a bus-based event, so it can't be watched from the CPU.

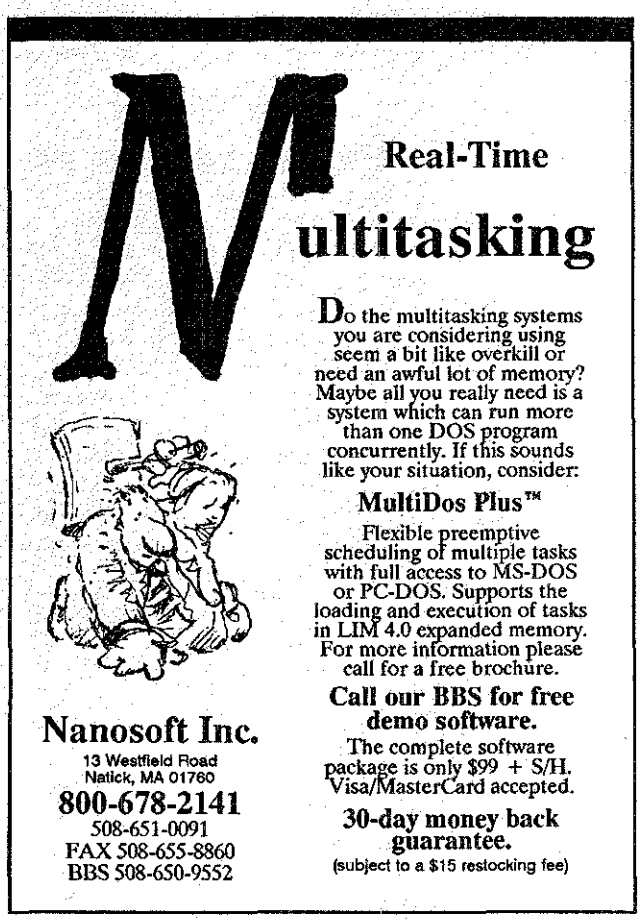
Current real-time products gather their information at the pins of the CPU. This works well on systems up through

ILLUSTRATION BY JIM CARPENTER

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LISTING 1

```
   #7: {
1AF9:0000 55            PUSH    BP
1AF9:0001 8BEC          MOV     BP,SP
1AF9:0003 B81000        MOV     AX,0010
1AF9:0006 9A6C02011B    CALL    __CHKSTK
1AF9:6AC8      0000  Write                                   [1F] -0046
1AF9:6AC6 1AF9       Write                                   [0E] -0044
1AF9:6AC4      000B  Write                                   [07] -0043


1B01:026C 59            POP     CX
1B01:026D 5A            POP     DX
1B01:026E 8BDC          MOV     BX,SP
1B01:0270 2BD8          SUB     BX,AX
1B01:0272 720B          JB      027F
1B01:0274 3B1E0401      CMP     BX,[STKHQQ]
1AF9:6AC4      000B  Read                                    [1F] -003F
1AF9:6AC6 1AF9       Read                                    [0A] -003E
1B01:0278 7205          JB      027F
1B01:027A 8BE3          MOV     SP,BX
1B01:027C 52            PUSH    DX
1B01:027D 51            PUSH    CX
1B01:027E CB            RETF
1AF9:5544      0FB0  Read STKHQQ                             [18] -003A
1AF9:6AB6 1AF9       Write                                   [0B] -0038
1AF9:6AB4      000B  Write                                   [07] -0037
1AF9:6AB4      000B  Read                                    [08] -0036
1AF9:6AB6 1AF9       Read                                    [08] -0035

   #10:     lower = 0;          /* lower limit of temperature table */
1AF9:000B C746F60000    MOV     WORD PTR [BP-0A],0000
   #11:     upper = 300;        /* upper limit */
1AF9:0010 C746F22C01    MOV     WORD PTR [BP-0E],012C
```



Real-Time
ultitasking
Do the multitasking systems you are considering using seem a bit like overkill or need an awful lot of memory? Maybe all you really need is a system which can run more than one DOS program concurrently. If this sounds like your situation, consider:
MultiDos Plus™
Flexible preemptive scheduling of multiple tasks with full access to MS-DOS or PC-DOS. Supports the loading and execution of tasks in LIM 4.0 expanded memory. For more information please call for a free brochure.
Call our BBS for free demo software.
The complete software package is only $99 + S/H. Visa/MasterCard accepted.
30-day money back guarantee.
(subject to a $15 restocking fee)
Nanosoft Inc.
13 Westfield Road
Natick, MA 01760
800-678-2141
508-651-0091
FAX 508-655-8860
BBS 508-650-9552


```
   #12:     step = 20;          /* step size */
1AF9:0015 C746F41400    MOV     WORD PTR [BP-0C],0014
1AF9:6ABE 0000       Write                                   [27] -0030
   #14:     fahr = lower;
```

LISTING 2

```
F000:D71F 90            NOP
0000:0004       3B  Int Ack, Probe                          [43] -0002
0000:0000       08  Int Ack, Probe                          [33] -0001
0000:0020 F000 FEA5  Read Int 08                            [12]  0000

F000:FEA5 1E            PUSH    DS
F000:FEA6 50            PUSH    AX
F000:FEA7 52            PUSH    DX
36C5:02E4      0246  Write                                  [15] +0002
36C5:02E2 F000       Write                                  [07] +0003
36C5:02E0      D720  Write                                  [06] +0004
36C5:02DE 0040       Write                                  [06] +0005
36C5:02DC      2267  Write                                  [06] +0006
36C5:02DA 0124       Write                                  [06] +0007
F000:FEA8 B84000        MOV     AX,0040
F000:FEAB 8ED8          MOV     DS,AX
F000:FEAD FF066C00      INC     WORD PTR [006C]
F000:FEB1 7504          JNZ     FEB7
0040:006C      6304  Read                                   [2B] +000C
0040:006C      6305  Write                                  [08] +000E

F000:FEB7 833E6E0018    CMP     WORD PTR [006E],+18
F000:FEBC 7513          JNZ     FED1
0040:006E 000F       Read                                   [24] +0012

F000:FED1 E9AAAD        JMP     AC7E

F000:AC7E A04000        MOV     AL,[0040]
F000:AC81 3CFF          CMP     AL,FF
F000:AC83 751F          JNZ     ACA4
F000:AC85 F6063F0007    TEST    BYTE PTR [003F],07
0040:0040        F0  Read                                   [36] +0019
F000:AC8A 7418          JZ      ACA4

F000:ACA4 FE0E4000      DEC     BYTE PTR [0040]
F000:ACA8 7520          JNZ     ACCA
0040:0040        F0  Read                                   [1E] +001D
0040:0040        EF  Write                                  [08] +001F

F000:ACCA CD1C          INT     1C
0000:0070 F000 FF53  Read Int 1C                            [25] +0022

F000:FF53 CF            IRET
36C5:02D8      0093  Write                                  [1E] +0024
36C5:02D6 F000       Write                                  [07] +0025
36C5:02D4      ACCC  Write                                  [06] +0026
36C5:02D4      ACCC  Read                                   [0B] +0027
36C5:02D6 F000       Read                                   [08] +0028
36C5:02D8      0093  Read                                   [06] +0029

F000:ACCC B020          MOV     AL,20    ; ' '
F000:ACCE E620          OUT     20,AL
F000:ACD0 5A            POP     DX
F000:ACD1 58            POP     AX
F000:ACD2 1F            POP     DS
F000:ACD3 CF            IRET
  0020          20  Out                                     [38] +002D
36C5:02DA 0124       Read                                   [08] +002E
36C5:02DC      2267  Read                                   [0A] +002F
36C5:02DE 0040       Read                                   [0A] +0030
36C5:02E0      D720  Read                                   [0D] +0031
36C5:02E2 F000       Read                                   [06] +0032
36C5:02E4      0246  Read                                   [06] +0033

F000:D720 74F8          JZ      D71A
```

PJ CODE

All code in this issue is available through our bulletin board services or directly from *PJ* on disk. For information see pages 72 and 73.

Circle 24 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

INTENSIVE. IN-DEPTH.

PACKED WITH C SOURCE CODE.

(& RIGHT NOW, FREE.)

DATA COMPRESSION • HASH TABLES • DOS INTERFACE • MICE • LASER PRINTERS • DEBUGGING • DATA EDITING ALGORITHMS • FONTS • GRAPHICS • TREES • BINARY SEARCH • TELECOMMUNICATIONS • AND MORE!

C Gazette is the only code-intensive C and C++ publication. Each issue offers cover-to-cover code with supporting text. Topics are treated in depth and presented in a clear, cogent manner that emphasizes real-world solutions.

For a **FREE preview copy,** fill in your name and address on the attached card. Or for faster service, call **1-800-234-0386.** We'll send you the next issue of ***C Gazette*** and invoice you for a one-year subscription. If you are not satisfied, write cancel on the invoice. The free issue is yours to keep.

1 year (6 issues) **$24.95**

49103

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

A STEP BEYOND.
PROMICE takes ROM emulation a step beyond. It's an affordable, multi-operational development tool with:
• on board intelligence
• modular design
• source level debugging
• future expandability
PROMICE, The Firmware Development System of Tomorrow...
Circle 25 on Reader Service Card

SUBSCRIBE!

Now at last . . . More Power, Speed and Performance in One Convenient, Affordable Package with . . .

The TesSeRact Programmer's Power Pak

Finally, there are C language programming tools available that let you work smarter, faster and more cost-effectively than ever before! The Power Pak is an all-in-one package that combines the power of the basic user interface with high-level, high-speed data compression; basic customized printer output and basic data file management.

At just $229, the TesSeRact Programmer's Power Pak provides more power for less money that competing products. And, best of all, there are no run-time royalties, and full source code is included!

Call or Fax your order today!

1-800-926-4551
1-215-443-9705
1-215-443-9753 (fax)

IDC Innovative Data Concepts
122 North York Road, Suite 5, Hatboro, PA 19040

Circle 26 on Reader Service Card

the 80486 because the address, data, status, and byte-enable signals are all available at the CPU pins. But the system register values are *not* available there, so breakpoints on register values can't be set in real time.

If the system's cache is external to the CPU, there is no effect on the real-time breakpoints because the cache is downstream from the point where the CPU information is gathered. The 80486 presents a problem, however: because it has an on-chip cache, memory accesses to cached memory never appear at the CPU pins. Other than using an expensive, full-blown ICE, the only option is to temporarily disable the cache. When the cache is disabled, all memory accesses are shown on the pins of the CPU so that an externally based device can see them.

A valuable qualifier for real-time breakpoints is a hardware-based pass counter. Using this capability, you can watch for the *N*th event before activating the debugger. Other helpful qualifiers are data and bit breakpoints, as mentioned above. With these, you can qualify memory or port breakpoints by their data access types and values. For example, you may wish to set a breakpoint when the happy face character appears on the color display.

Sequential triggering (or more correctly, a multiple-state state machine), which is a capability built into some real-time hardware products, lets you watch for one breakpoint. When it occurs, you switch immediately (in hardware) to watching for another breakpoint. For example, if you know that a particular memory location (BAR) is being overwritten by a particular function (FOO), you can set breakpoints that move from hardware state 0 to hardware state 1 on entry to FOO and then move from state 1 to state 0 on exit from FOO. A third breakpoint would watch for writes to BAR while in state 1 (but not in any other state!). When this event occurs, the debugger could be activated or another state could be invoked.

HISTORY IN THE MAKING

The real-time trace buffer is the system's execution history captured in a circular buffer. This information includes the CPU address, data, status, and byte-enable signals described above. The buffer's depth typically varies from 1 to 16K.

The buffer can be viewed in a number of ways, including the raw (or uncooked) mode, in which the address, data, and status fields are displayed as they are read from the buffer. Another mode reconstructs the instructions using only the instruction prefetch cycles. A mixed mode combines the two, showing the instructions along with the memory and the I/O port reads and writes performed by the instructions. When disassembling memory, the debugger shows your program's source code and symbols wherever possible.

To keep the processor from running out of instructions, the Intel family of processor chips have a prefetch queue that is refilled whenever it becomes empty or when a transfer of control—such as a jump, call, or interrupt—occurs. Thus, instructions may be fetched from memory but never actually executed. In order to decode the real-time trace buffer, the software must go to great pains to emulate the processor's actions and correctly intuit the flow of the program. In practice, the only significant problem is a conditional jump, because it can be hard to detect whether one has occurred.

The execution history captured in the buffer is extremely valuable, because it lets you see how your program got where it is. For example, if your program calls a subroutine as a NEAR procedure but the subroutine does a FAR return, an extra word is read from the stack by the FAR return and is used as the new code segment. This leaves an unbalanced stack and an incorrect execution location (the instruction pointer is correct, but the code segment is not). The program will probably execute an illegal instruction soon. Without a debugger that is aware of illegal instructions, the system would soon hang, giving no indication of what went wrong. With a debugger and a real-time trace buffer, however, the first illegal instruction executed would activate the debugger. You could then examine the real-time trace buffer to see what led up to the illegal instruction. Instead of having to guess what went wrong, you could see the fault condition.

BUFFER BENEFITS

Just as the presence of a resource tends to invite its use (remember how you thought your 80MB hard disk would

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

last forever?), having a real-time trace buffer lets you solve problems that were previously written off as "another system hang." In addition to the address, data, status, and byte-enable signals, the trace buffer can include external signals and CPU cycle-count information. An external signal can be used to watch a signal that is not normally captured, such as DMA or another external event. The CPU cycle-count information is the ultimate high-resolution timer, showing execution speed with 40-nanosecond resolution on a 25-MHz system. This level of detail can be useful when you're optimizing code.

In order to effectively lengthen the trace buffer or capture events that are widely spaced in time, the trace buffer can be set to capture only selected breakpoints or states. For example, if you want to capture the next 2000 port accesses to I/O ports in the range of 3F8h to 3FFh and nothing else, you can easily do so. If you want to exclude the capture of everything that happens in a particular hardware state, you can do that also. Because even a 16K trace buffer is filled up in approximately 5 milliseconds on a 25-MHz 80386, the effective lengthening of the trace buffer can be valuable.

Even when selective capture is not used, you can search for trace buffer entries that meet certain criteria. For example, you can search for all references to addresses that match a single specific value or a range of values. It is also possible to filter the display so that only one type of record, such as a memory write, is displayed.

Normally, the hardware breakpoint event is at the end of the trace buffer, and the entire trace buffer shows what happened before the debugger was activated. It's also useful to be able to set the trigger position to the center or top of the buffer. The center position is especially valuable because you can see what led up to a breakpoint and what happened afterward, all in real time. When the trigger position is set to the top, the first event in the buffer is the breakpoint event, and all other events show what happened after the breakpoint.

The real-time trace buffer provides a key ingredient—visibility. When you first discover a nontrivial bug, the cause is usually not obvious; you have to establish a hypothesis and then prove or disprove it. Being able to describe the problem in terms of hardware events, such as memory or I/O reads or writes, lets you capture the problem with real-time breakpoints and examine it using a real-time trace buffer. Instead of having to *guess* what may have happened, you can *see* what really occurred. Once you can see a problem's cause, its solution is often relatively easy.

TRACE BUFFER CONTENTS

Listings 1 and 2 show the contents of the Periscope Model IV real-time hardware trace buffer in the mixed mode. Listing 1 is the FTOC program from K&R, showing a trace from **_main** into the beginning of the program. Listing 2 is a system-timer tick (IRQ 0). For both listings, the first field is an address.

If a source-code line is displayed, the remaining fields are not shown on the source line; they can be displayed on the following line if desired. Both the source-code line and the assembly code are shown in Listing 1. If an assembly-level instruction is shown, the rest of the line contains the bytes that make up the instruction and the disassembly.

For other nonfetch cycles (for example, **Read**, **Write**, **In**, or **Out**), the second field is the data associated with the CPU event. The next field shows the cycle type, followed by a symbol name when appropriate. The number in braces to the right of the display indicates the CPU clock counts. The last field is the trace buffer sequence number.

IN SUMMARY

Hardware-assisted debuggers are a tremendous help when you need to understand and solve bugs that are beyond the scope of software-only debuggers. Although Intel is gradually building more debugging capability into the CPU, the current on-chip solutions don't provide enough power to help you solve difficult debugging problems. From the ability to get into a hung system via a break-out switch to the cycle-by-cycle detail provided by a real-time trace buffer, the visibility that hardware-assisted debuggers provide is a definite advantage. It can mean that you spend hours instead of days (or even weeks) tracking down those elusive bugs.

AMX

REAL-TIME MULTITASKING KERNEL

8086/88, 80x86/88 *80386*
Z80, 64180, 8080/85 *68000/10/20*

- Fast, reliable operation
- Compact and ROMable
- PC peripheral support
- DOS file access
- C language support
- Preemptive scheduler
- Time slicing available
- Configuration Builder
- Complete documentation
- Intertask messages
- Message exchanges
- Dynamic operations
  - — task create/delete
  - — task priorities
  - — memory allocation
- Event Manager
- Semaphore Manager
- List Manager
- InSight™ Debugging Tool

THE BEST

Join over 600 developers such as IBM®, Xerox, Hewlett Packard, Hayes, Hughes Aircraft and NASA.

CHOOSE AMX

The best low-cost, high-performance real-time multitasking system available today.

No Royalties
Source Code Included

Demo Disk	$25 US	*Call for prices for other processors.*
Manual only	$75 US	
AMX 86	$3000 US	

(Shipping/handling extra)

IBM is a registered trademark of IBM Corp.
Z80 is a trademark of Zilog, Inc.
AMX, AMX 86, InSight are trademarks of KADAK Products Ltd.

KADAK Products Ltd.
206-1847 West Broadway
Vancouver, B.C., Canada
V6J 1Y5
Telephone: (604) 734-2796
Fax: (604) 734-8114

Circle 27 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Object-oriented SCSI programming

Writing portable code to access SCSI devices using the Common Access Method

By Brett Glass

The ANSI-standard, high-speed I/O bus known as the SCSI (Small Computer System Interface) is supported by an unprecedented variety of computers and peripherals. If you own a Mac, a Sun, a NeXT, or an Amiga with a hard disk, you're probably already using SCSI, whether you know it or not. The same is true if you have a PC clone with a Bernoulli box, a CD-ROM, a magneto-optical disk, or a WORM drive.

Much of SCSI's popularity is connected with the success of Apple's Macintosh computers. The original Macs had a sealed case and only two relatively slow serial ports. It was possible to kludge in a hard drive, but the lack of peripheral slots made installing it an act of delicate, ad hoc surgery. Apple was reluctant to make the Mac significantly bigger or heavier by adding peripheral slots. (This had to wait for the NuBus-based Mac II.) Instead, they added a SCSI interface, which allowed several peripherals to be attached, daisy-chain fashion, to a single connector.

Because SCSI is the only way to build one product that can interface to virtually every Mac (slotted or not), Apple's choice has sparked a wealth of SCSI peripherals—including disk and tape drives, scanners, FAXes, network cards, laser printers, film recorders, and even graphics displays—that use SCSI. Most manufacturers' fastest and largest disk drives connect to the rest of the world through a SCSI interface.

Given the nearly universal support that SCSI has gained, you'd think that porting code among different platforms with SCSI devices would be easy. That hasn't been true, until now. A new standard, Common Access Method (CAM), has the potential to provide a great deal of portability.

FUZZY SCSI

The original ANSI SCSI standard, called SCSI-1, specified the hardware interface between a computer or peripheral and the SCSI bus. It also specified, albeit loosely, the types and formats of commands that could be sent to those devices. However, frustrated programmers quickly learned that SCSI-1 didn't specify anything at all about writing code to interface with those peripherals. (A tighter specification, the Common Command Set, will be included in the next version of the SCSI spec.)

Many system vendors have treated SCSI as if it were a single-purpose interface rather than a general-purpose bus. A Unix system, for example, typically has a driver that handles SCSI disks and perhaps a SCSI tape drive—and that's all. In most cases, that driver expects to "own" the computer's SCSI interface, making it infeasible to add another driver that provides support for a new kind of peripheral attached to the same interface. The only solution is to obtain the driver source, if you can, and hack in support for the new device.

Other systems offer better support for SCSI peripherals. Some make provisions for several device drivers to share the SCSI interface, making calls to a common set of low-level SCSI services. Most Sun workstations (except the 386i) have a set of functions called the Sun Common SCSI Architecture (SCSA). Unfortunately, on many of these operating systems you still need to write a device driver for each device, and very few peripheral makers can afford to do low-level kernel hacking for many systems.

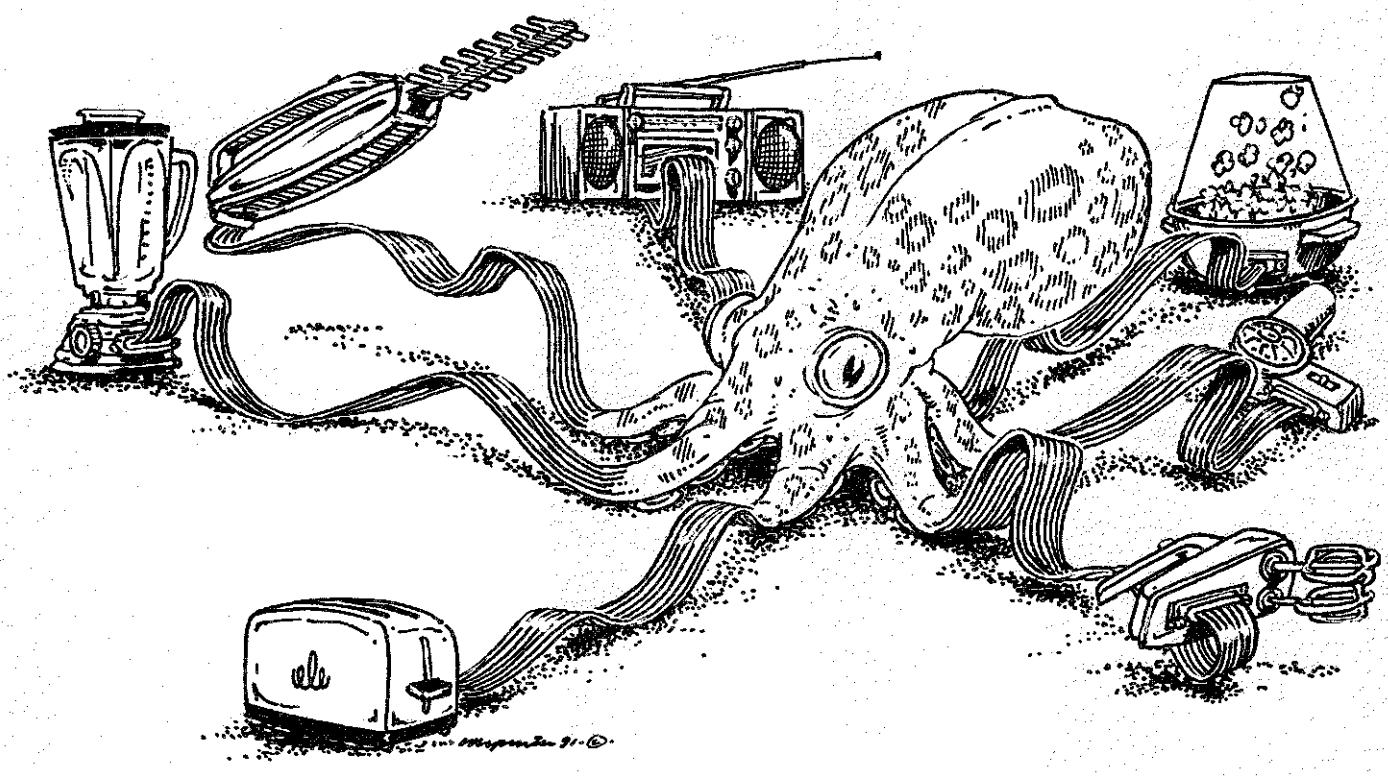
The best support is offered by systems that allow any application program to use a SCSI device. For example, Macintosh's SCSI Manager is part of the Macintosh Toolbox, so any program can use it. Because this interface is unique to the Macintosh, however, the code isn't portable.

IBM'S BACKPLANE BLUES

The most frustrating SCSI software incompatibilities of all occur in the IBM PC and its clones. Because IBM did not support SCSI devices on any of its PCs

Brett Glass is a free-lance writer, programmer, and consultant residing in Palo Alto, California. He can be contacted at rogue@well.sf.ca.us on Internet, on CompuServe at 72267,3673, or as glass on Bix.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



until the most recent PS/2 models, anarchy has reigned. Each vendor has provided its own SCSI interface, together with a DOS device driver that "takes over" that interface. A PC user with several different kinds of SCSI devices might end up with a backplane completely filled with SCSI cards—each servicing only one peripheral—instead of a single SCSI bus and interface handling many peripherals. What's more, preventing the cards from fighting for I/O port addresses, ROM addresses, and interrupts can be a nightmare.

Because IBM did not support SCSI devices on any of its PCs until the most recent PS/2 models, anarchy has reigned.

Another problem is that vendors often "hide" SCSI behind a hardware interface that emulates the original IBM PC/AT's ST-506 disk controller. Rather than face the problem of writing drivers for the dozens of operating systems that run on the PC, board vendors simply emulate the one disk interface that all of these systems support. Unfortunately, the SCSI interface is often so well hidden that the card can be used *only* for hard disks, contributing to the backplane explosion mentioned above.

Two vendors have stepped forward with proprietary solutions that let IBM PC programs and device drivers share a SCSI interface. Adaptec, which makes hard disk controller cards, has devised an Advanced SCSI Programming Interface (ASPI). Adaptec's set of APIs is very similar to Apple's SCSI Manager. So far, however, no other vendors have implemented ASPI.

When IBM began to use SCSI, it introduced its own proprietary interface, the Subsystem Control Block (SCB) architecture. Unfortunately, this scheme was best suited to buses like the Micro Channel (rather than the ubiquitous ISA). Vendors also feared that implementing it in hardware might infringe on IBM's pending patents. Finally, the details of the SCB architecture were released only after the CAM committee (see below) had nearly finished its work.

CAM TO THE RESCUE

The overwhelming problems of SCSI compatibility on the PC platform—and the difficulty of porting programs that used SCSI devices between platforms—prompted members of ANSI's SCSI committee to work on some standards of their own. In 1988, interface expert Dal Allen formed the Common Access Method (CAM) committee, which produced several working documents. One describes AT attachment (ATA) and Extended AT Attachment (EATA), two standards that nail down the details of making a SCSI controller emulate an AT disk controller. The most important document to come out of the CAM committee describes CAM, a universal, platform-independent API for SCSI programming.

Figure 1 (p. 34) shows a diagram of the CAM architecture. Note that device drivers as well as applications can communicate with the CAM transport (or XPT) layer, which serves as a dispatcher for SCSI requests. The XPT layer talks to the SCSI interface modules, or

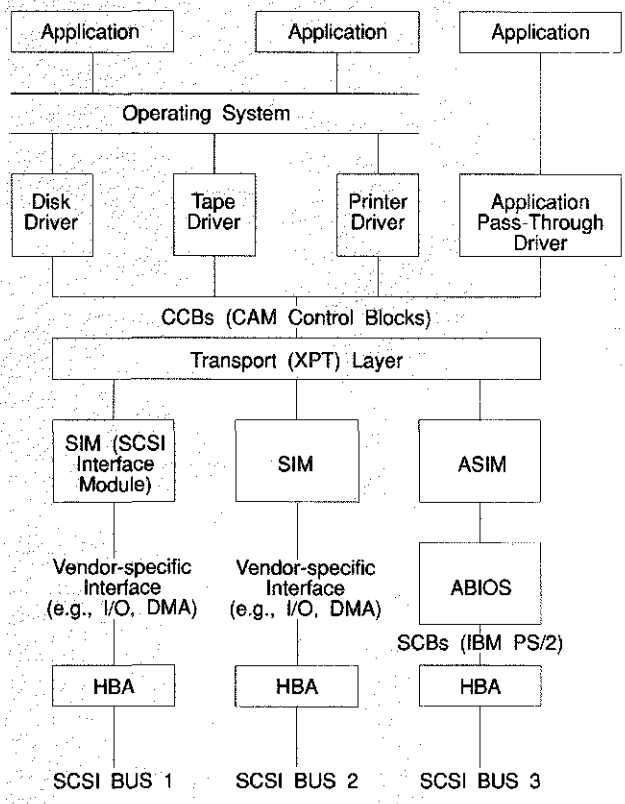
TABLE 1: A CAM control block

```
----------CCB Header----------
OS-Dependent (OSD) Fields:
  Address of this CCB
  Length of the whole CCB
  XPT function code
  Returned CAM subsystem status
  Filler bytes (if required)
Common Fields:
  Path ID (identifies which SCSI bus
    to use)
  Target device ID
  Target logical unit number
  Multipurpose flag field
-------CCB-Dependent Fields--------
Many possible OSD and common
  fields, as required by the func-
  tion being performed
```

ILLUSTRATION BY JIM CARPENTER

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

FIGURE 1: The CAM architecture



LISTING 1

```
program CAMTest;
{ This program performs a simple test to see what SCSI devices are
  attached to an MS-DOS machine with a CAM driver. The CAM unit tests
  for the presence of CAM automatically. This program then performs
  a Path Inquiry command -- which verifies that a SCSI bus exists and
  that the program has allocated sufficient buffer space. It then
  sends a series of Device Inquiry commands to each of the 8 possible
  SCSI device addresses and reports what it finds. The CAM unit provides
  an object-oriented framework for this process and hides most of its
  complexity. }
{ Copyright (C) 1991 by Brett Glass, All Rights Reserved.
  Original program concept from BallardSynergy CAM Developer's Toolkit,
  Copyright (C) 1990 by BallardSynergy, All Rights Reserved.
  Used with permission. }
uses CAM;
var
  pathInq : PathInquiryCCB; {CCB for path inquiry request}
  scsiReq : SCSIRequestCCB; {CCB for SCSI I/O requests}
  inqBuf  : array [0..31] of Byte.; {Buffer for data returned by in-
                                     quiry}
  id : Byte; {SCSI ID loop counter}
  i : Integer; {Temporary character counter}
procedure WriteDevType(d : Byte);
{ Write the type of device that's associated with a number, according
  to the SCSI spec }
begin
case d of
  0 : Write('Disk');
  1 : Write('Tape');
  2 : Write('Printer');
  4 : Write('WORM')
else
  Write('Unknown')
  end;
end;
begin {Main Program}
{Doing a Path Inquiry is as simple as invoking two methods.}
with pathInq do
  begin
  Init; {Set all fields to their default values}
  Submit; {Submit the CCB for processing}
  {Wait for completion; pro forma on single-tasking systems like MS-DOS}
  while camStatus = STAT_REQUEST_IN_PROGRESS do;
  if camStatus <> STAT_REQUEST_DONE_NO_ERROR then
    begin
    Writeln('Path Inquiry returned error ', camStatus);
    Halt;
    end;
  if privateDataSize > SCSI_REQUEST_FILL_SIZE then
    begin
    Writeln('Error: CCB not big enough to allow SCSI requests');
    Halt;
    end;
  end;
Writeln('SCSI Device Report');
Writeln('------------------');
for id := 0 to 7 do {Loop over all 8 SCSI ids}
  begin
  with scsiReq do
    begin
    Init;
    camFlags := CAM_DIR_DATA_IN; {We're receiving data}
    targetID := id; {From this device}
    cdbLength := 6; {We're sending 6 bytes of command}
    dataLength := 32; {And receiving 32}
    dataPtr :=  @inqBuf; {In this buffer}
    cdb[0] := $12; {Inquiry command}
    cdb[4] := 32; {Maximum size of buffer}
    Submit; {Submit the request}
    while camStatus = STAT_REQUEST_IN_PROGRESS do; {Wait for completion}
    Write(id, ': ');                               {Report the results}
    case camStatus of
      STAT_REQUEST_DONE_NO_ERROR:
        begin
        {A device is there; say what kind it is and show its ID string}
        WriteDevType(inqBuf[0]);
        Write(' -- ');
        for i := 8 to 32 do
          Write(Char(inqBuf[i])); {Write string returned by inquiry}
        Writeln;
        end;
      STA.T_SELECTION_TIMEOUT:
        Writeln('No device present');
      STAT_SCSI_BUS_BUSY:
        Writeln('Device busy');
      STAT_INVALID_TARGET_ID:
        Writeln('Host')
    else
      Writeln('** ERROR **')
      end;
    end;
  end;
end.
```

LISTING 2

```
unit CAM;
{$A-}
{ This Turbo Pascal 6.0 unit is an object-oriented interface to
  BallardSynergy's CAMpatible SCSI drivers, Version 1.302. Please note
  that the CAM specification is still in flux so these bindings may
  need to change to accommodate changes in the interface. }
{ Turbo Pascal bindings Copyright (C) 1991 by Brett Glass,
  All Rights Reserved.
  Original C bindings, sample code, and assembly language
  Copyright (C) 1990 by BallardSynergy, All Rights Reserved.
  Used with permission. }
interface
function CAMPresent : Boolean; { Return TRUE if a CAM driver is in-
stalled }
{ This enumerated type gives the CAM function codes for the
  BallardSynergy implementation  of CAM, version 1.302.
  The numbering is different from that found in the CAM UNIX
  bindings and in Microsoft's LADDR spec. }
type CAMFunctionCode = (
  FUNC_UNUSED_00,
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

SIMs—software drivers that control each of the SCSI interfaces attached to the machine. For ABIOS on the PS/2, a special SIM, called the ASIM, maps CAM requests to IBM's proprietary SCB architecture.

CLASS AND SUPERCLASS

The CAM Control Block (CCB) is the key to the CAM architecture. This layered data structure passes information back and forth between the driver or application and the system's lower layers. The CCB's structure has only a few fields whose sizes are OS- and CPU-dependent, but the header files for different operating systems should be able to hide most of the differences.

Table 1 shows the anatomy of a generic CCB. Note that although the header is the same for all CCBs on a system, the "CCB-dependent fields" vary, depending on the request being made.

The layered nature of CCBs suggests an especially apt way to implement CCBs in object-oriented languages such

Turn your favorite C compiler into a powerful database manager with the

C/Database Toolchest

The C/Database Toolchest™ adds sophisticated file management functions to your Power C™, Turbo C®, QuickC®, or Microsoft® C compiler. With the C/Database Toolchest™, your data requires much less disk space than with programs like dBASE®, and you can access your data much faster. Of course the full power of C provides you with an unlimited amount of programming flexibility.

The C/Database Toolchest™ includes three major components:

1) An advanced B+tree library gives you instant access to your data.

2) A high-level ISAM library provides you with an easy-to-use C interface, and

3) A complete database manager (with C source code included) shows you how to create impressive applications.

You also receive a comprehensive 350 page manual and a utility for converting dBASE® files.

The C/Database Toolchest™ supports features that you'd expect to find only in products costing ten times as much. Advanced features include variable length records, variable length keys, multiple keys per index, and multiple indexes stored in a single file. Your data files can contain an unlimited number of records, and each record can be as large as 32K bytes in length.

About the only thing that the C/Database Toolchest™ doesn't do is cost you a lot of money. We've kept our price low so you can manage your budget as easily as your data.

Now Only $19.95!

Order Coupon

Name_____
Street_____
City_____
State_____ Zip_____
Telephone_____
Paying By____ Money Order____Check
____ Visa ____ MC ____ AX ____ Disc.
Card#_____
Exp. Date_____
Disk Size____ 5¼" ____ 3½"

Qty.	Product	Price	Subtotal
____	C/Database Toolchest	$19.95	____
____	C/Database Library Source. B+tree & ISAM library source code	$10.00	____
	Add Shipping ($5 USA, $20 Foreign)		____
	Texas Residents Add 8% Sales Tax		____
	Total Amount of Your Order		____

Order now by calling our toll free number or mail the coupon to:

Mix Software
1132 Commerce Drive
Richardson, TX 75081

1-800-333-0330

60 Day Money Back Guarantee
Not Copy Protected ■ Royalty Free
For technical support, please call:
1-214-783-6001

C/Database Toolchest and Power C are trademarks of Mix Software. QuickC and Microsoft C are registered trademarks of Microsoft. Turbo C is a registered trademark of Borland. dBASE is a registered trademark of Ashton-Tate.

if (U == Microsoft®_ISV
&& U.GUI == Windows™(3)
&& (U.SDK || U.MDK || U.DDK)
&& U.VAL > MinWages)

Softelligence™ libraries minimize the time spent porting DOS programs or developing new apps for Windows 3. Maximize your power using the libraries optimized for the new multimedia & desktop publishing standards:

MICROSOFT WINDOWS
Version 3.0 Compatible Product

"More Power To You!"™ lets you control Windows 3 at the DPMI (DOS Protected Mode Interface) level. Instantly access video & other hardware for animation, arcade-style games, & real-time processing. You decide when to control hardware, & when to let the GDI control it. DOS graphics run with minimum modifications, & much faster than GDI graphics.

"More Memory For You!"™ gives you the quickest possible port from the DOS c/malloc functions, & a fast virtual memory allocator for small & medium memory models. Also includes many "lstr/lmem" functions missing from the SDK.

Animagic 256™ reads & simultaneously displays &/or animates &/or color-cycles multiple multimedia standard 256-color 320 x 200 GIF™ &/or Animator™ FLI files. Requires *"More Power To You!"* described above.

Bezier Master™ is believed to be the fastest software implementation for bezier curve display, invented at Softelligence, using only 16-bit integers & no multiplies or divides! Bezier curves are the most important curves today; PostScript™ is completely based on them.

Type Master™ reads, decrypts & decodes Adobe™ Type 1 PostScript fonts, stores all data including subroutines and hints, & even translates to a Type 3 font. Reads the hundreds of commercial PostScript fonts blazingly fast! Includes functions to display font outlines that interface to *Bezier Master*.

Call Softelligence, **(800)446-8088**, or FAX, (206)284-3898. $395 per library. Multi-product discounts & source code available. MC/Visa accepted. No royalties.

(C)1991 Softelligence, 3213 W Wheeler St, Suite 140, Seattle WA 98199. Microsoft is a registered trademark & Windows is a trademark of Microsoft Corp. *"More Power To You!"*, *"More Memory For You!"*, *Animagic 256*, *Bezier Master* & *Type Master* are trademarks of Softelligence. All other trademarks are trademarks of their respective companies.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
  FUNC_ABORT_SCSI_COMMAND,
  FUNC_EXECUTE_SCSI_IO,
  FUNC_GET_DEVICE_TYPE,
  FUNC_UNUSED_04,
  FUNC_PATH_INQUIRY,
  FUNC_RELEASE_SIM_QUEUE,
  FUNC_RESET_SCSI_BUS,
  FUNC_RESET_SCSI_DEVICE,
  FUNC_SET_ASYNC_CALLBACK,
  FUNC_SET_DEVICE_TYPE,
  FUNC_SET_HBA_PARAMETERS,
  {Create dummy names to reserve unused function numbers}
  FUNC_UNUSED_0C, FUNC_UNUSED_0D, FUNC_UNUSED_0E, FUNC_UNUSED_0F,
  FUNC_UNUSED_10, FUNC_UNUSED_11, FUNC_UNUSED_12, FUNC_UNUSED_13,
  FUNC_UNUSED_14, FUNC_UNUSED_15, FUNC_UNUSED_16, FUNC_UNUSED_17,
  FUNC_UNUSED_18, FUNC_UNUSED_19, FUNC_UNUSED_1A, FUNC_UNUSED_1B,
  FUNC_UNUSED_1C, FUNC_UNUSED_1D, FUNC_UNUSED_1E, FUNC_UNUSED_1F,
  FUNC_CHANGE_OS2_NAME,
  FUNC_ROM_DEBUGS);

type
  SCSIid = 0..7;

{ This abstract superclass is the "father" of all classes of CAM
  Control Blocks (CCBs). }

type C.CB = object
  ccbLen : Word;     {Length of CAM Control Block (CCB)}
  funcCode : CAMFunctionCode;   {XPT Function Code}
  camStatus : Byte;  {Returned CAM subsystem status}
  scsiStatus : Byte; {Returned SCSI status}
  pathId : Byte;     {Adapter to do request}
  camFlags : Longint; {Flags containing transaction parameters}

  {Note: CCB.Setup is called by the constructors of the child
   classes of CCB, but is *not* a constructor itself. If
   it were, a VMT pointer would be allocated between the
   fields of this parent class and the fields of child classes,
   causing everything to be misaligned. It's OK for the
   child classes to have constructors. }

  procedure Setup(func : CAMFunctionCode; len : Word);
  procedure Submit;
  end;

type CCBAddr = ^CCB;

{ Constants for the CAM flags in the BallardSynergy implementation. These
  are different from the UNIX definitions that appear in the CAM spec.}

const
  CAM_DATA_DIRECTION              = $C0000000;  { BITS 30 & 31 }
  CAM_DISABLE_AUTOSENSE           = $20000000;  { BIT 29 }
  CAM_DATAPTR_IS_SG_LIST_PTR      = $10000000;  { BIT 28 }
  CAM_DO_NOT_CALLBACK             = $08000000;  { BIT 27 }
  CAM_LINKED_COMMAND              = $04000000;  { BIT 26 }
  CAM_QUEUE_ACTION_SPECIFIED      = $02000000;  { BIT 25 }
  CAM_CDB_FIELD_IS_CDB_PTR        = $01000000;  { BIT 24 }
  CAM_DO_NOT_ALLOW_DISCONNECT     = $00800000;  { BIT 23 }
  CAM_INIT_SYNC_TRANSFERS         = $00400000;  { BIT 22 }
  CAM_DISABLE_SYNC_TRAN           = $00200000;  { BIT 21 }
  CAM_CDBPTR_IS_PHYS_ADDR         = $00004000;  { BIT 14 }
  CAM_DATAPTR_IS_PHYS_ADDR        = $00002000;  { BIT 13 }
  CAM_SENSEPTR_IS_PHYS_ADDR       = $00001000;  { BIT 12 }
  CAM_MSGPTR_IS_PHYS_ADDR         = $00000800;  { BIT 11 }
  CAM_LINKPTR_IS_PHYS_ADDR        = $00000400;  { BIT 10 }
  CAM_CALLBACKPTR_IS_PHYS_ADDR    = $00000200;  { BIT  9 }
  CAM_DATA_BUFFER_VALID           = $00000080;  { BIT  7 }
  CAM_STATUS_VALID                = $00000040;  { BIT  6 }
  CAM_MESSAGE_BUFFER_VALID        = $00000020;  { BIT  5 }
  CAM_RESERVED_BITS               = $001F811F;  { BITS 1-4,8,14-20 }

  CAM_DATA_DIRECTION_CLEAR        = $3FFFFFFF;  { BITS 30 & 31  }

  CAM_DIR_DATA_IN                 = $40000000;
  CAM_DIR_DATA_OUT                = $80000000;
  CAM_DIR_NO_DATA                 = $C0000000;

  DATA_IN  = 1;
  DATA_OUT = 0;

{ Status codes returned by CAM }

const
  STAT_REQUEST_IN_PROGRESS          = $00;
  STAT_REQUEST_DONE_NO_ERROR        = $01;
  STAT_ABORTED_BY_HOST              = $02;
  STAT_UNABLE_TO_ABORT              = $03;
  STAT_COMPLETE_WITH_ERROR          = $04;
  STAT_CAM_BUSY                     = $05;
  STAT_INVALID_REQUEST              = $06;
  STAT_INVALID_PATH_ID              = $07;
  STAT_SCSI_DEVICE_NOT_INSTALLED    = $08;
  STAT_WAIT_FOR_TIMEOUT             = $09;
  STAT_SELECTION_TIMEOUT            = $0A;
  STAT_COMMAND_TIMEOUT              = $0B;
  STAT_SCSI_BUS_BUSY                = $0C;
  STAT_MESSAGE_REJECT_RECIEVED      = $0D;
  STAT_SCSI_BUS_RESET               = $0E;
  STAT_UNCORRECTED_PARITY_ERROR     = $0F;
  STAT_REQUEST_SENSE_FAILED         = $10;
  STAT_NO_HBA_DETECTED_ERROR        = $11;
  STAT_DATA_OVERRUN_OR_UNDERRUN     = $12;
  STAT_UNEXPECTED_BUS_FREE          = $13;
  STAT_PHASE_SEQUENCE_FAILURE       = $14;
  STAT_CCB_LENGTH_INADEQUATE        = $15;
  STAT_CANNOT_PROVIDE_CAPABILITY    = $16;
  STAT_INVALID_LUN                  = $20;
  STAT_INVALID_TARGET_ID            = $21;
  STAT_FUNCTION_NOT_IMPLEMENTED     = $22;
  STAT_NEXUS_NOT_ESTABLISHED        = $23;
  STAT_INVALID_INTIATOR_ID          = $24;
  STAT_INVALID_DATA_BUFFER          = $25;

{ Values for flags that appear at top of status code }

const
  FLAG_SIM_QUEUE_FROZEN = $40;
  FLAG_AUTOSENSE_DATA_VALID = $80;

const
  INQUIRY_DATA_LEN = 36; {Length of data returned from device
                          inquiry. Defined by SCSI-1 and unlikely
                          to change in the future...}

type
  InquiryData = array [0..Pred(INQUIRY_DATA_LEN)] of Byte;
  InquiryDataAddr = ^InquiryData;

type AbortXPTRequestCCB = object(CCB)
  ccbPtr : CCBAddr;
  constructor Init;
  end;

type GetDeviceTypeCCB = object(CCB)
  targetID : SCSIid;
  lun : Byte;
  deviceType : Byte;
  inquiryDataPtr : InquiryDataAddr;
  constructor Init;
  end;

type SetDeviceTypeCCB = object(CCB)
  targetID : SCSIid;
  lun : Byte;
  deviceType : Byte;
  constructor Init;
  end;

type ResetSCSIBusCCB = object(CCB)
  constructor Init;
  end;

type ResetSCSIDeviceCCB = object(CCB)
  targetID : SCSIid;
  hostStatus : Byte;
  targetStatus : Byte;
  constructor Init;
  end;

type ReleaseSIMQueueCCB = object(CCB)
  targetID : SCSIid;
  lun : Byte;
  constructor Init;
  end;

const
  SIM_ID_LEN = 16;
  HBA_ID_LEN = 16;
  VU_HBA_LEN = 16;

type
  SIMid = array [0..Pred(SIM_ID_LEN)] of Char;
  HBAid = array [0..Pred(HBA_ID_LEN)] of Char;
  VUhba = array [0..Pred(VU_HBA_LEN)] of Char;

const
  FEAT_SOFT_RESET_SUPPORTED          = $01;
  FEAT_COMMAND_QUEUING_SUPPORTED     = $02;
  FEAT_LINKED_COMMANDS_SUPPORTED     = $08;
  FEAT_SYNCHRONOUS_TRANSFER_SUPPORTED = $01;
  FEAT_SCSI_BUS_SUPPORTED_16_BIT     = $02;
  FEAT_SCSI_BUS_SUPPORTED_32_BIT     = $04;
  FEAT_RELATIVE_ADDRESSING_SUPPORTED = $08;

type PathInquiryCCB = object(CCB)
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

as C++ and Turbo Pascal. If the header fields are defined as part of an abstract superclass, each of the types of CCB can be defined as a class that "inherits" from the superclass and adds additional fields for each request. The resulting SCSI transactions can set their own defaults and can be instructed to "submit themselves" on request.

CHECKING THE SCSI BUS

I was able to test this concept thanks to BallardSynergy, a Cupertino, California, company that supplies CAM drivers to many SCSI OEMs. After connecting an NCR SCSI interface and a 32MB Seagate hard drive to my lab computer, I learned how to put the drive through its paces and created a set of SCSI bindings I could use in my daily programming.

Listing 1 is a simple test program for CAM that takes inventory of the devices on a SCSI bus. Two kinds of CAM control blocks—a path inquiry block and a SCSI I/O request block—are used to gather this information. The path inquiry ensures that the SCSI bus is attached and that the program has provided sufficient working space for the CAM driver. The SCSI I/O request sends a device inquiry command to each of the eight possible bus addresses.

Listing 2 shows the part of the program that does (and conceals) most of the work required to use CAM. This CAM unit, written for Turbo Pascal 6.0, automatically sets up the CCB objects, tests the CAM installation, and dispatches requests to the CAM driver. It uses Turbo Pascal features such as the in-line assembler, object-oriented extensions, and strong typing (which helps prevent bugs). The CAM driver is invoked by calling INT 4Bh with a pointer to the CCB in ES:BX. A similar module would be easy to implement in any object-oriented language, such as Modula-3 or C++.

GAINING MOMENTUM

The CAM standard is now close to completion and is being worked on by ANSI's XT39.2 standards group. They plan to finish the standard by June 1991, but this date may slip further.

Still, hardware and software vendors aren't waiting for SCSI-2's CAM to be set in stone before implementing it on their systems. Although demands from regular customers may prompt them to continue to support older, proprietary standards, most vendors have shown an interest in providing CAM support as well. You can expect to see CAM become the lingua franca of SCSI interfaces by the middle of this decade, eliminating once and for all the problems of using SCSI on different platforms. DJ

RESOURCES

- For more information on CAM and SCSI, call the SCSI BBS at (316) 636-8700 (300/1200/2400). Maintained by John Lohmeyer of NCR, chairman of the SCSI Committee, this BBS has drafts of the latest versions of the proposed ANSI specifications available for downloading.
- For information on CAM drivers or the CAM Developers' Toolkit, contact BallardSynergy at (408) 257-8844.

Circle 211 on Reader Service Card

Resource Converter

Translate your resource files from **PM to Windows,** or **Windows to PM**. Save weeks of trial and error converting them yourself. Get the job done right the first time with the **Resource Converter!**

Converts:

Dialog Templates
All Control types
Std. Control ID's
Menus & Tables

PM to Win converter now shipping for $225, Win to PM shipping soon.

Petrasoft
3665 East Bay Drive Suite 204-112
Largo, Florida 34641
Msg: 813-530-9481, FAX: 813-530-9264

All trademarks or tradenames used here-in are owned by their respective companies.

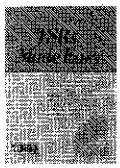
Circle 52 on Reader Service Card

TOOLS FOR TURBO 6.0

Write powerful Turbo Pascal programs faster and easier using these tools.

Object Professional
A comprehensive object-oriented library provides user interfaces, data objects, and systems routines.
COMPUTER LANGUAGE PRODUCTIVITY AWARD 1990
Only $189.

B-Tree Filer
A database toolbox for powerful network applications.
Only $189.
Single-user version, $139.



TSRs Made Easy
TSR routines from Object Professional at an economical price.
Only $49.

Turbo Professional
A non-OOP library of more than 600 powerful routines. The predecessor of Object Professional.
Only $125.

Turbo Analyst
Nine analytical tools in an integrated programming environment.
Only $129.

Call toll-free to order
1-800-333-4160

9AM–5PM PST M-F, US & Canada

All tools include full source code, complete documentation, and CompuServe and telephone support direct from the authors. You pay no royalties.

TURBO Power

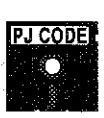
TurboPower Software
P. O. Box 66747 Scotts Valley, CA 95067-0747

Add $5 per order for standard shipping US & Canada. Dual media (except TPro-5.25" only). For more information call (408) 438-8608, Fax (408) 438-8610, CIS 76004,2611.

Circle 30 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
  featureList : Longint;
  highestPathID : Byte;
  initiatorID : SCSIid;
  simVendorName : SIMid;
  hbaVendorName : HBAid;
  vendorUnique : VUhba;
  privateDataSize : Longint;
  OSD : Pointer;
  constructor Init;
  end;

type SetHBAParametersCCB = object(CCB)
  featureList : Longint;
  constructor Init;
  end;

type Proc = Procedure;
type ProcPtr = ^Proc;

type AENFlag = (
  AEN_UNSOLICITED_SCSI_BUS_RESET,
  AEN_UNSOLICITED_RESELECTION,
  AEN_RECOVERED_FROM_PARITY_ERROR,
  AEN_SCSI_ASYNCRONOUS_EVENT_NOTIFY);

type AENFlagSet = set of AENFlag;

type AsyncCallbackCCB = objec.t(CCB)
  targetID : SCSIid;
  lun : Byte;
  aenFlags : AENFlagSet;
  callbackPtr : ProcPtr;
  constructor Init;
  end;

const
  CDB_LEN = 16;

type CDBType = array [0..Pred(CDB_LEN)] of Byte;

const
  SCSI_REQUEST_FILL_SIZE = 113;

type SCSIRequestCCB = object(CCB)
  targetID : SCSIid;
  lun : Byte;
  queueAction : Byte;
  vendorFlags : Word;
  cdbLength : Word;
  senseLength : Word;
  messageLength : Word;
  sgListLength : Word;
  dataLength : Longint;
  timeOut : Longint;
  dataPtr : Pointer;
  sensePtr : Pointer;
  messagePtr : Pointer;
  linkPtr : CCBAddr;
  peripheralPtr : Pointer;
  callBackPtr : ProcPtr;
  unused : array [1..2] of Pointer;
  cdb : CDBType;
  filler : array [1..SCSI_REQUEST_FILL_SIZE] of Byte;
  constructor Init;
  end;

implementation

uses DOS; {For type Registers}

procedure CCB.Setup(func : CAMFunctionCode; len : Word);
begin
FillChar(self, len, 0); {Zero out the CCB}
funcCode := func;       {Set the function code and length}
ccbLen :.= len;
end;

procedure CCB.Submit;
var
  regs : Registers;
begin
regs.bx := Ofs(self);
regs.es := Seg(self);
regs.ax := $C3D4; {Sentinel}
Intr($4B,regs);
end;

constructor AbortXPTRequestCCB.Init;
begin
CCB.Setup(FUNC_ABORT_SCSI_COMMAND, Sizeof(self));
end;

constructor GetDeviceTypeCCB.Init;
begin
CCB.Setup(FUNC_GET_DEVICE_TYPE, Sizeof(self));
end;

constructor SetDeviceTypeCCB.Init;
begin
CCB.Setup(FUNC_SET_DEVICE_TYPE, Sizeof(self));
end;

constructor ResetSCSIBusCCB.Init;
begin
CCB.Setup(FUNC_RESET_SCSI_BUS, Sizeof(self));
end;

constructor ResetSCSIDeviceCCB.Init;
begin
CCB.Setup(FUNC_RESET_SCSI_DEVICE, Sizeof(self));
end;

constructor ReleaseSIMQueueCCB.Init;
begin
CCB.Setup(FUNC_RELEASE_SIM_QUEUE, Sizeof(self));
end;

constructor PathInquiryCCB.Init;
begin
CCB.Setup(FUNC_PATH_INQUIRY, Sizeof(self));
end;

constructor SetHBAParametersCCB.Init;
begin
CCB.Setup(FUNC_SET_HBA_PARAMETERS, Sizeof(self));
end;

constructor AsyncCallbackCCB.Init;
begin
CCB..Setup(FUNC_SET_ASYNC_CALLBACK, Sizeof(self));
end;

constructor SCSIRequestCCB.Init;
begin
CCB.Setup(FUNC_EXECUTE_SCSI_IO, Sizeof(self));
end;

function CAMPresent : Boolean;
{ This function tests for the presence of a CAM driver. It's
  taken directly from the code in BallardSynergy's CAM Development
  Kit. Turbo's inline assembler makes it unnecessary to do the
  assembly as a separate step. }
label
  no_cam;
begin
CAMPresent := FALSE;
asm
  xor ax,ax
  mov es,ax
  mov bx,$012C
  les bx,dword ptr es:[bx]
  add bx,8
  cmp byte ptr es:[bx],'S'
  jne no_cam
  cmp byte ptr es:[bx+1],'C'
  jne no_cam
  cmp byte ptr es:[bx+2],'S'
  jne no_cam
  cmp byte ptr es:[bx+3],'I'
  jne no_cam
  cmp byte ptr es:[bx+4],'_'
  jne no_cam
  cmp byte ptr es:[bx+5],'C'
  jne no_cam
  cmp byte ptr es:[bx+6],'A'
  jne no_cam
  cmp byte ptr es:[bx+7],'M'
  jne no_cam
  mov byte ptr  @Result,TRUE
no_cam:
  end;
end;

begin {Module init code}
if not CAMPresent then
  begin
  Writeln('CAM .driver not present in system');
  Halt
  end;
end.
```



All code in this issue is available through our bulletin board services or directly from *PJ* on disk. For information see pages 72 and 73.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The i860 Has Arrived!

Nine years ago, MicroWay used a similar headline to proclaim the arrival of the Intel 8087. In 1982, our runtime libraries made it possible to use an 8087 with the IBM FORTRAN, Pascal and BASIC compilers. Since then we have sold hundreds of thousands of Intel math coprocessors. We think that the i860 is every bit as significant a product as the 8087 and it will help to change the way supercomputer power gets distributed to the desktop. A single i860 is 1600 times as fast as an 8087, 4 to 10 times as fast as a 486/4167 and twice as fast as a Cray 1! And, if that is not enough, we recently achieved 90% efficient parallel operation and 135 Megaflops using two MicroWay Number Smasher-860s.

MicroWay has again taken the lead in introducing an emerging numerics technology. Since August, we have sold hundreds of Number Smasher-860s to our loyal customers who have utilized them to do everything from designing refineries to predicting the weather.

One reason Number Smasher-860 has been so well accepted is that it uses our NDP compilers to drive the i860. These compilers include features that make porting from the 386 or VAX a snap, including VAX/VMS extensions to NDP Fortran and MS C extensions to NDP C. We also offer a growing list of tools that make it easier to take advantage of the i860 (see descriptions below).

How upward compatible are our languages? Each NDP-860 language comes with an i860 DOS Extender that makes it possible to call MS DOS directly from an i860 application. Our GREX-860 graphics library includes the MS C graphics routines as a subset and automatically drives all popular video adapters. If you are reluctant to run an 80 megaflop processor under MS-DOS, we can provide device drivers and interfaces that let our board run the three popular flavors of UNIX or run on a UNIX network as a computation server. If you prefer DOS, like most of our customers, we have versions of our extender that are compatible with both DESQview and Windows 3.0. To sum it up, about the only excuse you have for not switching to a Number Smasher-860 is that you either enjoy waiting to get results back or you get a thrill spending the hundreds of dollars per day it costs to rent a 3090 or Cray.

For more information, please call MicroWay's Technical Support Dept. at (508) 746-7341.

386, 486 and i860 Compilers

Our NDP family of compilers generate globally optimized, mainframe quality code that runs on the 386, 486 or i860 in protected mode under UNIX, XENIX, SunOS, or extended DOS. The compilers address 4 gigabytes of memory while supporting the 80287, 80387, Weitek, and Cyrix coprocessors. Applications can mix code from all four compilers and assembly language. To simplify your ports, we offer ClearView, our full-featured, windowing symbolic debugger that works with DOS versions of NDP 386 and 486 compilers.

NDP Fortran™ is a full F77 with F66 and DOD extensions that is 99% VMS compatible. Also contains new F90 features.

NDP C™ runs in two modes—K&R with Sys V and MS C extensions or 100% ANSI as validated by Plum Hall.

NDP Pascal™ is a full ANSI/IEEE Pascal, with extensions from C and BSD 4.2 Pascal.

NDP C++™ is a fully AT&T v.2.0 compatible C++ compiler (not a preprocessor) that contains a full ANSI C compiler as a C++ subset.
NDP C++ 386/486 ..... $1195

NDP Fortran, C or Pascal
860 versions include toolset and run on
DOS, UNIX or XENIX $1995
386 DOS versions include NDP Tools $895
486 DOS versions include NDP Tools $1195
UNIX/XENIX 386/486 versions ..... CALL
NDP VMEM Virtual Memory Manager $295

GREX 386/860 — A Library of 160 lower level graphics routines (free with DOS compilers): $495
ClearView™ 3/486 Symbolic Debugger: $395
NDP-CrossRef™ Our full function cross reference utility that produces comprehensive data and function listings. Available for all NDP compiler versions. $195
NDP Windows™ Library: $125, C Source: $250
NDP Plot™ $325
NDP/FFT™ NDP or 80x87 version ... ea. $250
Halo Professional $595

NDP NAG™ Workstation library is a subset of the NAG mainframe libraries. Has 172 routines for differential equations and eigenvalue problems, matrix operations, fitting curves, statistics and regression analysis ..... 386 Version: $895
T800 NAG for Transputers $2750

NumberSmasher-860™

Number Smasher-860 is the highest performance coprocessor card ever to run in an ISA or EISA bus or as part of a transputer system. Delivers up to 80 million floating point operations per second at 40 MHz and produces over 12 double precision Linpack megaflops. The board comes standard with an ISA interface, 8 meg of high speed memory, two transputer link adapters, and your choice of NDP Fortran, C++, C or Pascal for the i860 running under MS-DOS, UNIX, XENIX or SunOS ..... 33 MHz: $6995
40 MHz: $9200 32 MB version: add $4000

NDP NAG/860 is a Microway port of the NAG workstation library to the i860 ..... $1495
NDP HALO-860 includes HALO Professional and an interface library which makes HALO directly callable by the i860 ..... $995
860 Vectorizer includes PSR VAST-2 which converts a FORTRAN program into a new program which calls the Intel Vector library. Includes the Intel Vector Library and MicroWay's 860 Profiler $1495
860 Vector Library — Written for Intel, the library includes over 400 vector primitives that take advantage of the i860's cache, dual instruction mode, and pipelined multiplier accumulator $495

SX Your AT!

Microway's FASTCache-SX was PC Week's "Analyst's Choice" for AT SX accelerators. It is the ideal upgrade to run Windows 3.0 on your AT as it has outperformed all SX motherboards that have been tested to date.
FASTCache-SX™ — The most cost effective accelerator on the market today. This ¾-length 16-bit card plugs into your I/O channel and 286 socket, speeding up all applications by a factor of 2 to 4. Runs all 386 applications, OS/2 and Windows 3.0. Features a 16 or 20 MHz 80386-SX, a 4-way 32K cache (expandable to 64K) and a math coprocessor socket.
16MHz: $495 20 MHz: $595 Cable: $95

FASTCache-SX/Plus™ — The "Windows Solution" — combines the FASTCache-SX accelerator with up to 8 megabytes of extended memory. The Plus occupies a 16 bit slot and can be optionally purchased with memory and/or a coprocessor. The 0K prices are:
16 MHz: $595 20 MHz: $695 Cable: $95

Parallel Processing

MicroWay's IBM compatible Monoputer, Quadputer, Videoputer, and Linkputer boards work together using Inmos transputers to provide expandable, plug-in mainframe performance for your desktop PC.

Monoputer™— Includes one T800 and up to 16 meg of RAM for parallel code development. The 4 MWhetstones T800 makes it the ideal FORTRAN engine for cost-effective execution of your mainframe programs. ..... from $1295

Quadputer™— This board for the AT or 386 can be purchased with 1 to 4 transputers and 1 or 4 meg of memory per transputer. Two or more Quadputers can be linked together to build networks of up to 100 or more transputers providing mainframe power ..... from $1845

Linkputer™— Links up to 8 boards to provide dynamic transputer topologies $1500

NumberSmasher-486™

Personal Workstation magazine, June 1990, said, "The Number Smasher-486 lives up to its name. Its numeric performance exceeds that of all 25-MHz systems we've tested to date. It gives you top 486 performance for the best price."
Number Smasher-486™ — Microway's family of 486 motherboards includes ISA and EISA models running at 25 and 33 MHz. All take advantage of 486 Burst mode and the top of the line includes a 256K second level cache, and SIMM sockets for up to 64 megabytes of RAM, and a 4167 socket. ..... from $2495

Math Coprocessors

WEITEK, INTEL, CYRIX

4167-25/33 ..... $995/$1295			
3167-20/-25/-33 ..... $495/ $795/ $995			
mW3167 Micro Channel-25/33 .. from $1295			
mW3167/80387 Board $200			
8087 ..... $80		**8087-2** $115	
80387-16SX $280		**80387-20SX** .. $300	
80387-16 .. $295		**80387-20** $345	
80387-25 .. $439		**80387-33** $540	
80287XL .. $180		**287Turbo-20™** $300	

Cyrix SX16MHz: $230 **SX 20 MHz**: $255
20 MHz: $279 **25 MHz:** $359 **33 MHz:** $449

MicroWay

World Leader in PC Numerics

Corporate Headquarters: P.O. Box 79, Kingston, MA 02364 USA (508) 746-7341
32 High St., Kingston-Upon-Thames, U.K. 081-541-5466
USA FAX (508) 746-4678 Germany 069 75 2023 Holland 40 836455 Italy 02-74 90 749 Japan 81 033 222 0541

Contact this advertiser directly.

PRODUCED BY COMPUTERMUSEUM.ORG.NZ
ELECTRONIC REPRODUCTION PROHIBITED

Numeric exception handling

Demystifying the IEEE 754/854 floating point standards, denormals, and NaNs

By Mark J. Barrenechea

Numeric exception handling is a facet of compiler runtime support that is not well understood by many application programmers, or even by some compiler vendors.

In a general sense, a numeric exception is a numeric error, like a division by zero or taking the square root of a negative number. The job of detecting and reacting to exceptions is handled differently by various numeric processors. Usually, the job is shared by the runtime libraries, the exception handler, and the processor.

In proactive programming, runtime libraries handle trig functions such as **sin** and **cos** by avoiding exceptions and range problems instead of by letting them occur, as is the case with reactive programming. The exception handler senses traps and provides correct results by, for example, rounding. The processor may be able to manipulate denormal numbers and other anomalies automatically.

For example, with its NPX-Status word-enabled, the Intel 387 performs denormal arithmetic on the co-processor without the need for any vendor-supplied software. Denormal arithmetic need not be part of a runtime library or an exception handler, because it is provided by the 387's hardware. However, Intel's i860 does not perform denormal arithmetic, and it generates an exception on all i860 instructions that contain a denormal as a source operand. For the i860, denormal arithmetic needs to be part of an exception handler.

Most MS-DOS compilers come with a default exception handler that gets linked into every bound executable. This handler can be removed or replaced by the programmer. It is usually part of the start-up routine that is traditionally called **crt0**. Unix and Xenix do not allow exception handlers, because the operations they perform are usually restricted to the kernel.

This article discusses the issues surrounding exception handlers under 32-bit protected-mode MS-DOS. Examples are drawn from both the Intel 387 and i860 families of processors. But first, let's look at the ANSI/IEEE-854 standards for the representation of numbers. Denormal arithmetic is also part of this discussion because, in my view, it is essential for any marketable exception handler.

Mark J. Barrenechea is director of language development at MicroWay, Inc., in Plymouth, Massachusetts.

FLOATING-POINT STANDARDS

In 1977, Intel adopted a floating-point number format for its FPAL and subsequent iSBC 310 and 8232 processors. In 1979, the Institute for Electrical and Electronic Engineers (IEEE) proposed a new industry standard for microcomputer floating-point arithmetic in the historic paper, "A Proposed Standard for Binary Floating Point Arithmetic," by Coonen et al. [Reference 1]. The standard, which had already been adopted by Intel, was designed to promote portability and a uniform programming environment between computer systems. This format is imperative to the discussion of numeric anomalies such as exceptions. The concept embodied in the Coonen paper became the basis for IEEE floating standard 754.

IEEE-Std-854 (1987) [2], the latest IEEE standard, defines a commercially feasible way for computer systems to perform floating-point arithmetic. Although its use is voluntary, all popular processors conform to it. IEEE-854 is a generalization of ANSI/IEEE-Std-754 (1985), which removes dependencies on radix (number base) and word length. IEEE-854 and IEEE-754 are used interchangeably in this article.

The IEEE-854 addresses the following areas:

- constraints on parameters that define values of basic and extended floating-point numbers;
- add, subtract, multiply, divide, square root, remainder, and compare operations;
- conversions between integers and floating-point numbers (casting);
- conversions between different floating-point precisions (single, double, etc.);
- conversions between basic precision floating-point numbers and decimal strings; and

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

- floating-point exceptions and their handling, including non-numbers (NaNs).

The IEEE-854 excludes the following:

- formats for internal storage of floating-point numbers (such as the Intel representation);
- encoding of integers and formats of strings of characters representing decimal numbers;
- interpretation of the sign and significand fields of NaNs; and
- conversions between extended precision floating-point numbers and decimal strings.

One interesting exclusion is the format of floating-point numbers. The 387 and i860 processors have defined this format for 32-, 64-, and (in the 387 only) 80-bit reals. The standard defines a range of constraints that must be supplied by a conforming implementation.

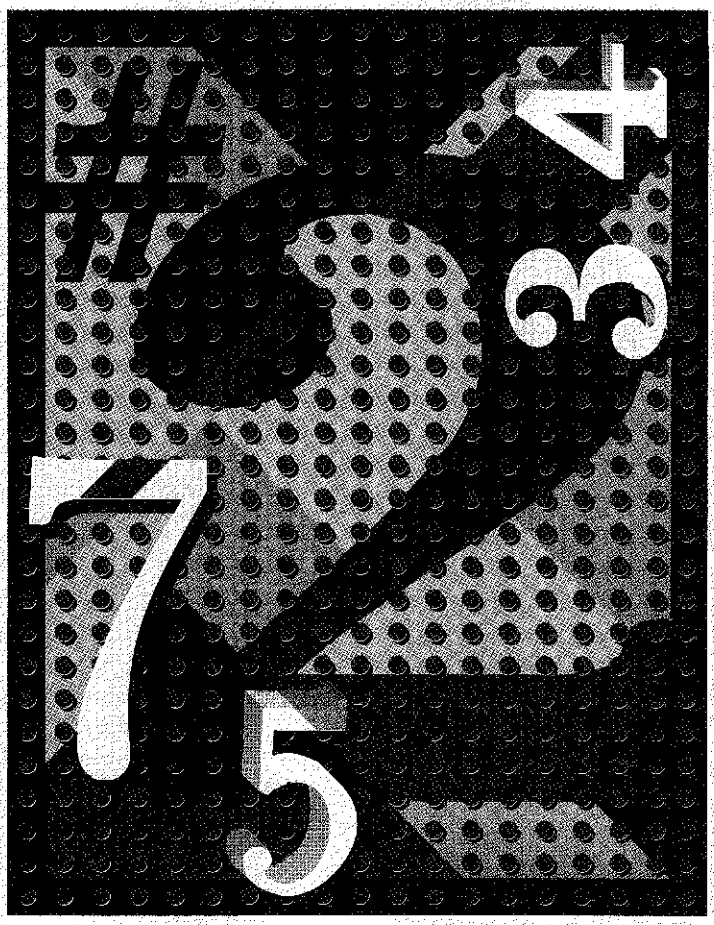


ILLUSTRATION BY ROB SCHUSTER

Chip manufacturers often make the misleading claim that they conform 100% to IEEE-854. An implementation objective of IEEE-854 was to allow an "implementation of a floating-point system conforming to this standard to be realized entirely in software, entirely in hardware, or in any combination of software and hardware." For both the Intel 387 and i860 processors, IEEE-854 arithmetic is realized in a combination of software and hardware. It is the "environment" that the end user sees that conforms or fails to conform to IEEE-854. For example, the i860 does not have a divide instruction, and division must be approximated using a Newton-Rhapson technique. Thus, the i860 processor by itself does not conform to IEEE-854. With the divide instruction supplied in software, the environment the end user sees can be made to conform with that portion of the standard.

Consider an abstract from Intel's i860 64-bit *Microprocessor Programmer's Reference Manual*, pp. 1–4 [3]:

"The floating-point data types, floating-point instructions, and exception handling all support the IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985) with both single- and double-precision floating-point data types. Due to the low-level instruction set of the i860 microprocessor, not all functions defined in the standard are implemented directly by the hardware. The i860 microprocessor supplies the underlying data types, instructions, exception checking, and traps to make it possible for software to implement the remaining functions of the standard efficiently. Intel offers a software library that provides programs for the i860 microprocessor with full IEEE-compatible arithmetic."

REPRESENTATION OF NUMBERS

Understanding exceptions is inseparable from understanding the IEEE-854 floating-point representation of numbers. Let's consider the practical problem of calculating the resistance of a network of parallel resistors. The calculation is determined by the following algebraic formula:

$$R_t = 1 / (1/R_1 + 1/R_2 + 1/R_3)$$

When any of the components in the network is zero, the total resistance of the network is also zero. Normal floating-point number systems would not handle this case because it requires that division by zero generate an infinity and that a subsequent division by infinity generate a zero. In a normal numeric system, there are zeros but no infinity, and dividing by zero is always an invalid operation. As we will discover, the numbers *plus-zero* and *minus-zero* are special (that is, they have their own special representation), as are the numbers plus-infinity and minus-infinity.

The existence of special numbers at both extremes of the number system (instead of at zero only) sets the IEEE number system apart from older systems used by micros and mainframes. In addition to signed zeros and infinities, the special numbers include denormals, which have a reduced precision; the NaNs, which are combinations of bits that fall outside the domain of the valid numbers and are intended to be used for special purposes; and a special NaN called an *indefinite real*. The Intel 387 produces the correct result for the above resistance problem by masking the divide-by-zero and precision exceptions. The Intel i860 needs special software supplied by the compiler vendor.

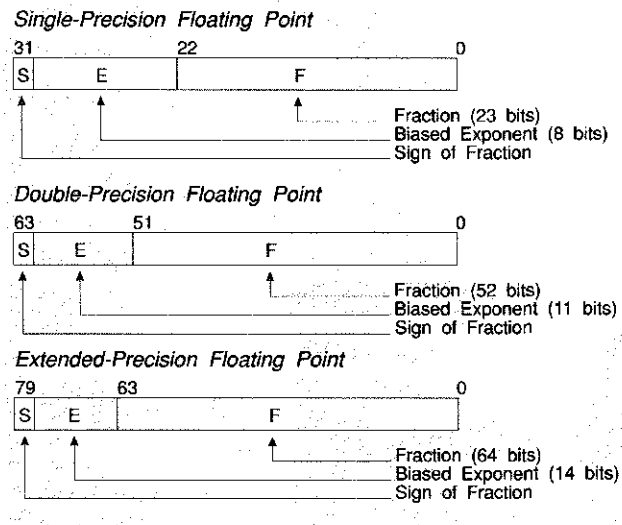
The binary floating-point numbers used by the IEEE convention cannot be used to represent any arbitrary number; that is, they do not have infinite precision. Generally, numbers near zero can be more precisely expressed than can very large numbers. In fact, the number of expressible real numbers between 0 and 1 is almost equal to the number of expressible real numbers between 1 and infinity.

The Intel formats for single-, double-, and extended-precision floating-point numbers are shown in Figure 1.

The single-precision floating-point data type uses a 32-bit

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

FIGURE 1: Formats for floating-point numbers



binary floating-point number, whereas a double-precision floating-point data type uses 64 bits. Double-precision floating-point numbers are more accurate, but calculations tend to be slower because of the added "bits."

This number system combines three binary components: a sign bit, a binary exponent/mantissa consisting of 8, 11, or 15 bits, and a binary significand/fraction consisting of 23, 52, or 63 bits. For example, the decimal floating-point number +4.0 can be expressed as the product of –1 raised to the power of the sign bit times a significand of 1.0 times 2 raised to the power of 2.

Single-precision floating-point numbers have a 23-bit significand and an 8-bit exponent. Thus, single-precision numbers have a range from 10^{-37} to 10^{+38} with a precision of at least six significant digits.

Consider a simple formula for deciphering single-precision floating-point numbers:

$$(-1)^{sign} \cdot (1.B_1B_2B_3, \ldots, B_{23}) \cdot 2^{(E-127)}$$

where *sign* is a 1 or 0 stored in the sign field, $B_1 \ldots B_{23}$ are the significand bits, and E is the unsigned value stored in the exponent field.

As an example, let's examine what happens with the exponent of a single-precision number. Single-precision numbers have an 8-bit exponent, so 256 values can be represented. The most obvious tactic would be to map one of these 256 values to an exponent of zero, leaving 255 for positive and negative exponents. But what about special values? Exponent values for zero and 255 are special, and thus reserved, leaving 1 . . . 254 to represent all "ordinary" exponents: negative, zero, and positive. One of this 254-member set must be used to represent an exponent of zero, leaving an asymmetric set of 253 members.

The IEEE committee chose to have 126 negative exponents and 127 positive exponents. Thus, the exponent zero is represented using the value 127 (0x7F), the values between 1 and 126 map to negative exponents, and the numbers between 128 and 254 map to positive exponents. The value 127, known as the bias, is added to the true exponent in order to obtain the "correct" value.

Figure 2 shows all floating-point encodings. Double-precision floating-point numbers have a 52-bit significand and an 11-bit exponent, with a bias of 1023. Double-precision floating-point numbers have a range from 10^{-307} to 10^{+308}, with 15 significant digits.

The following formula can be used to decipher double-precision floating-point numbers:

$$(-1)^{sign} \cdot (1.B_1B_2B_3, \ldots, B_{52}) \cdot 2^{(E-1023)}$$

In the Intel 387, temporary reals are 80-bit reals. Internally, the 80387 holds all numbers in this format; this extended range and this precision make the 387 more accurate than the i860. The fact that numbers are held in this temporary real format is transparent to the user.

DENORMAL ARITHMETIC

Denormals are what the IEEE-854 calls "subnormal" or "tiny" numbers. These special numbers fall between the smallest representable number and zero. This concept of valid but unrepresentable numbers may seem odd; understanding denormals means understanding precision. Let's consider nonnegative single-precision reals/floats and their range over the artificial and finite computer number system. The mantissa in this case is 24 bits: an implicit leading bit that is always 1, and 23 explicit bits that can be either 0 or 1. Therefore, the interval between any two powers of 2 contains just 2^{23} unique significands—that is, the number space between 1 and 2 can be divided into 8,388,608 unique real numbers, and similarly, there are approximately 8,388,608 unique real numbers between ½ and 1.

The number of reals stays constant in any interval, so there are approximately 8.38 million numbers between 1 and 2; 8.38 million numbers between 2 and 4; 8.38 million numbers between 4 and 8; and so on. Likewise, there are approximately 8.38 million real numbers between 1 and ½, between ½ and ¼, between ¼ and ⅛, and so on. Consequently, the density per interval grows between successively smaller negative powers of 2 and diminishes between cardinals in each interval between successively greater powers of 2. Thus, we have only 4.18 million reals between 2 and 3, because the same number of significands must be stretched over a greater range. In the interval from 4 to 8 there are about 2.09 million reals between cardinals, and from 8 to 16, about 1.04 million.

To understand the denormal, we must investigate what happens as the number gets smaller. This is equivalent to asking what happens as the IEEE-854 representation approaches zero. We start by taking the smallest valid single real/float, $1.0 \cdot 2^{-126}$, and divide it by 2 (see Table 1). This smallest number has a biased exponent of 1, which drops to 0 when it is divided by 2. An exponent of all zeros is used as a flag to indicate that the exponent has now passed its smallest value (–126) and that the system is now reducing the significand by factors of 2 (see Figure 1). The process begins by moving the first (implicit) significant binary digit to the right of the decimal point. The special numbers between the smallest number and zero no longer have an implied 1 at the head of their significand. As the

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Tear Your C Development Time in Half.

We understand the pressure you're under, trying to create, test, and debug complex C programs without Instant-C™, the professional development environment. With all your searching for errors and recompiling, you barely have time to test code, never mind read magazines.

Instant-C can make your time twice as productive. Here's how:

Industrial-strength debugging. Instant-C finds bugs other compilers miss, automatically detecting over 500 errors, including:

- *Array references out of bounds*
- *Stack overflow*
- *Use of uninitialized pointers*
- *Overwriting a return address*
- *Read or write through null pointers*
- *Overwriting your program or DOS*

Ultra-fast, incremental compiling. When you make a change to a program, Instant-C recompiles just the change. In fact, Instant-C can rebuild a 100,000 line program in less time than it takes to read our headline. And in recent tests, Instant-C was 24 times faster than Microsoft Programmer's WorkBench.*

Microsoft® C compatibility. Instant-C links with Microsoft C libraries to form an enhanced development environment. After developing, testing, and debugging with Instant-C, you can use Microsoft C to make the final version of your program.

Unlimited program size. Instant-C uses Rational Systems' DOS/16M technology, the software that enabled Lotus to break the 640K barrier for 1-2-3® Release 3. Because it uses extended memory, no program is too big, and your work isn't slowed down by swapping to disk.

That's not the half of it. Instant-C is a complete development environment, including source-level debugging, dynamic cross referencing, lint-like static check, support for third-party libraries, and interactive testing of single functions or partial programs.

The Instant-C Environment

- The industry's fastest edit-compile-link-debug cycle
- Configurable editor and source formatter
- Incremental compiler and linker
- Direct access to 16MB of memory
- Extensive static and run-time error checking
- Multiple virtual or real screens
- Source-level debugging
- Support of commercial libraries
- Conditional breakpoints in C
- Compatibility with assembly language object modules
- Immediate evaluation of C expressions, including macros
- Compatibility with Microsoft C compiler 5.1 and 6.0
- Interactive execution of functions
- Allows use of Microsoft run-time libraries
- Dynamic cross-referencing and browsing
- Full graphics screen support

Get the whole story. To find out more about how Instant-C can double your productivity, give us a call. We'll give you all the details in a flash. After all, we know how valuable your time is.

Rational Systems, Inc.

Rational Systems, Inc.
220 N. Main St.
Natick, MA 01760
Toll-free: (800) 666-2585
Outside U.S.: (508) 653-6006
Fax: (508) 655-2753

* Tests involved changing one line of code in a 11,500-line program, running on a 20MHz 80386 system with 8MB memory. Instant-C requires an 80286/386/486 and at least 1M of memory. Instant-C is a trademark of Rational Systems, Inc. Microsoft is a registered trademark of Microsoft Corp. 1-2-3 is a registered trademark of Lotus Development Corp.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

FIGURE 2: Floating-point encodings

		Class	Sign	Biased Exponent	Significand *ff–ff**
POSITIVES	NANS	Quiet	0 0	11..11 : 11..11	11..11 : 10..00
		Signaling	0 0	11..11 : 11..11	01..11 : 00..01
		Infinity	0	11..11	00..00
	REALS	Normals	0 0	11..10 : 00..01	11..11 : 00..00
		Denormals	0 0	00..00 : 00..00	11..11 : 00..01
		Zero	0	00..00	00..00
NEGATIVES		Zero	1	00..00	00..00
	REALS	Denormals	1 1	00..00 : 00..00	00..01 : 11..11
		Normals	1 1	00..01 : 11..10	00..00 : 11..11
		Infinity	1	11..11	00..00
	NANS	Signaling	1 1	11..11 : 11..11	00..01 : 01..11
		Quiet	1 1	11..11 : 11..11	10..00 : 11..11
* Integer bit is implied and not stored.			Single: Double: Extended:	< 8 bits> <11 bits> <15 bits>	<23 bits> <52 bits> <64 bits>

values are progressively halved, the significand bits continue to be shifted right until the number system runs out of room.

When the process finally shifts the last significant digit out of the fraction, it arrives at the IEEE-854 representation of positive zero, which is a zero-biased exponent and a zero significand. This process is called *gradual underflow*. In effect, the number system trades precision for range by turning off normalization for the sequence of numbers between the smallest number and zero.

There are 8,388,607 ($2^{23}-1$) of these positive single real denormals, providing a "cushion" between the smallest valid number and zero. Because you cannot halve your value and keep it too, the system must give up something with the introduction of denormals, and it gives up precision. Denormals extend the low-end range of single reals from 2^{-126} to 2^{-149}, but they lose a bit of precision for every power of 2 that they drop, until at the low end the precision of the significand is reduced to a single bit, then to zero.

The Intel 80387 does not automatically perform denormal arithmetic. Consider a single- or double-precision floating number as an operand to a floating-point instruction. This operand will be converted to a normalized 80-bit real, the operation will occur, and then the result will be "denormalized" back into the result precision. The 80387 does not do denormal arithmetic on 80-bit reals, because it cannot normalize an 80-bit denormal.

However, because the Intel 80860 has only a 64-bit real and not an 80-bit real, the 80860 needs to flush all denormals to zero, losing this "cushion" of 8,388,607 numbers. This is called *abrupt underflow*.

Why are denormals important? A denormal by itself is not necessarily significant. They are usually the result of an intermediate calculation, with the final result being a normalized number. Consider dividing 1 by a denormal:

$$1.0 / (1 \cdot 2^{-127})$$

On the Intel 80387, this operation produces a representable (but very large) number. On the Intel 80860, it generates a divide-by-zero exception. Thus, it is important to allow denormals to propagate when they enter a calculation, instead of their being flushed to zero to preserve program correctness.

BASIC EXCEPTIONS

Section 7 of the standard covers exceptions. Five basic exceptions are defined:

- invalid operand
- division by zero
- overflow
- underflow
- inexact results.

When each of these exceptions is encountered, it is signaled. Signaling consists of setting a status flag, executing a trap, or both. Each exception should be associated with an additional user-controlled trap. This means that the "environment" must provide a mechanism for providing correct results under these five exceptions, but it must also allow this mechanism to be replaced by the end user. IEEE-854 offers no well-defined precedence among simultaneous exceptions.

All this raises some major issues. First, and by default, a trap—whether originating in hardware or software—must deliver a correct response to a given operation. Setting status flags may be part of delivering the correct response. Second, the user must be able to disable and replace (and restore) the default exception handler. Disabling all exceptions, flushing all

TABLE 1: Gradual underflow

Number	Sign	Biased Exponent	Implied Bit	Significand
2^{-126}	0	1	1	00000000000000000000000
2^{-127}	0	0	0	10000000000000000000000
2^{-128}	0	0	0	01000000000000000000000
2^{-129}	0	0	0	00100000000000000000000
2^{-130}	0	0	0	00010000000000000000000
. . .	.	.	.	. . .
2^{-149}	0	0	0	00000000000000000000001
zero	0	0	0	00000000000000000000000

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

anomalies to zero, and terminating program execution when an anomaly occurs is *not* an IEEE-conforming environment. Crashing and burning on a divide-by-zero is not acceptable.

INVALID OPERATION EXCEPTIONS

This exception usually indicates a program error. The *invalid operation exception* is signaled if the operand(s) is invalid for a particular operation. The result is a quiet NaN, lacking a trap. The invalid operations are [1]:

- any operation on a signaling NaN
- addition or subtraction: magnitude subtraction of infinities such as (+INF) + (–INF)
- multiplication: 0 · (INF)
- division: (INF)/(INF) or 0/0
- remainder: x REM y, where y is zero or x is infinite
- square root if the operand is less than zero
- conversion of an internal floating-point number to an integer or to a decimal string when overflow, infinity, or NaN precludes a faithful representation in that format, and when this cannot otherwise be signaled
- conversion of an unrecognized input string
- comparison via predicates involving "<" or ">," without "?," or when the operands are "unordered" (e.g., NaNs).

DIVISION BY ZERO

If the divisor is zero and the dividend is a finite nonzero constant, then an exception is signaled. The result of this operation is a correctly signed (INF). That is, the sign is determined using normal rules (–/– = +; +/– = –; –/+ = –; +/+ = +).

If the exponent of a resultant arithmetic operation is too large for the destination format, an overflow is signaled. Conversely, if the exponent is too small, an underflow is signaled.

Overflows can occur when a result is rounded such that it would exceed the largest finite number in the destination format. Underflows can occur under two conditions: when the creation of a result may cause an exception in a subsequent operation, such as overflow upon division; and when the extraordinary loss of accuracy due to denormals (very tiny numbers) involved in the calculation occurs. Because the i860 does not do denormal arithmetic but does trap when an operand is a denormal, its floating-point status register allows underflows to become zeros when the appropriate bit is set, avoiding the trap on underflows. The 387 does denormal arithmetic, so underflows are rare.

INEXACT RESULTS AND EXCEPTION PRECEDENCE

The inexact exception occurs if the result of an operation is not exactly representable in the destination format. For example, 1.0/3.0 cannot be precisely represented in any binary format of finite precision. If the rounded result of an operation is not exact or if it overflows without an overflow trap, then an exception is signaled. The result is assigned either to the destination of the operation or to the trap handler. The inexact exception can occur often; it generally indicates that some precision, even though acceptable, is lost.

At times, multiple exceptions can occur on a given operation. IEEE-854 defines an exception precedence that is rather thin and thus deficient. If enabled, overflow and underflow take precedence over a simultaneous inexact trap. Let's consider an exam-

WINDOWS APPLICATIONS IN 1/10th TIME!

* YOU CAN RAPIDLY PROTOTYPE FOR WINDOWS 3.0!
* Generates C Code for Complete SAA CUA User Interfaces.
* Does DATA VALIDATIONS and Spreadsheet-like TABLES.
* Ports In Minutes to DOS,Windows 3.0 and OS/2.
* Easy to Learn. Be MORE Productive In a few hours.
* FRONT ENDS to 3270/Other Hosts,LAN,SQL Server,Sybase, Informix,Oracle,BTrieve,dBVista,CIndex,Paradox,dBase....

Instant Prototyper. Both DOS & Windows,only $99.
PROTOTYPES:fancy,data oriented,50+Windows in one week. ANIMATES:Windows/dialogs,Menus,Fields,Tables,Text File Editor,Help,Icons,radio/check/push buttons,pick/lists/combo boxes,Scroll bars,Bitmaps. VALIDATES:Custom/cross/lookup, display,required,picture(A9!LXB#Z$),range,date,time,money...

Instant Windows. Windows 3.0 $495.DOS $249.
Includes Prototyper,50+Function Library,350p Manual,C Code Generator (just add your DBMS/LAN/Internals) and examples.

Instant Windows Combo Pack. $995.
Includes Prototyper,Instant Windows for DOS & Windows,DLL, Linker,Dynamic Forms,250+Functions,1 Yr.Maintenance/Gold Level Tech.Support,4 hrs Premium Consulting,600p Manual.

CALL 415-324-9552 NOW! No Royalties.
No Runtimes.QUALITY Product,Tech.Support and Consulting.
WinSoft.1016 E.El Camino Real#216,Sunnyvale CA 94087.

Circle 33 on Reader Service Card

PROFESSIONAL SERIAL COMMUNICATION MADE EASY

DON'T BE FOOLED BY THE PRICE

COMM-DRV is a professional serial communication library that comprises a complete serial I/O library, a true MSDOS RS232 device driver, an RS232 TSR, a serial port monitor, and a dumb terminal emulator. It is second to none!!!

VERY EASY TO USE !

- 25+ serial ports sharing one or more IRQs. Remap COM ports to any address.
- Supports speeds up to 115.2 kilobaud.
- Hardware/Software Handshake - RTS/CTS DSR/DTR XON/XOFF
- Installable as a device driver or as a TSR (can be de-installed from RAM).
- Can be linked into application or used via TSR or device driver.
- Supports adjustable communication buffer sizes on every port.
- Runs under Desqview, Omniview, VPIX, Windows 3.0, and most multitaskers.
- Interface to DOS via INT21, INT14, C's open(), read(), write(), close(), ioctl().
- Selectively interrupt user routine on reception, transmission, & status change.
- Supports most multiport cards including Digiboards & Intel Hub card.
- Coexistence of different multiport, singleport, & smart cards.
- Supports the FOSSIL interface used on most bulletin boards (e.g. OPUS)
- Supports all file transfer protocols.
- 8250, 16450, 16550(including FIFO buffers), etc. support.
- Compatible with all languages(C,QBASIC,PASCAL,COBOL,etc.)
- Customization provided upon request.

Driver,Library,Sources $89.95
Driver,Library,Sources, & 4 PORT SERIAL CARD $199.95
Windows 3.0 Driver For Multiport Cards $99.95

TEL.(713)498-4832 FAX (713)-568-3334
1-800-966-4832 BBS (713)-568-6401
VISA/MC/AMEX/PO/COD/CHECK 24 Hours/Day 7 Days/Week
Willies' Computer Software Co.
2470 S. Dairy Ashford, Suite 188 Houston, Tx. 77077

Circle 34 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LISTING 1

```
if (isqnan(nom) || isqnan(dem) || issnan(nom) || issnan(dem)){
   set_psr(DIV_BY_NAN);
   result.i = QNAN;
   return result;
   }
else if (iszero(nom) && iszero(dem)){
   set_psr( DIV_BY_QNAN );
   result.i = QNAN;
   return result;
   }
else if (iszero(dem)) {
   set_psr( DIV_BY_ZERO );
   result.i = INF;
   return result;
   }
else if (isinf(nom) && isinf(dem)) {
   set_psr( INVALID_OP );
   result.i = QNAN;
   return result;
   }
result.f = float(nom.f) / float(dem.f);
return result;
...
```

LISTING 2

```
#include <iostream.h>
#include <string.h>
#include <stdlib.h> // for exit() prototype

const unsigned int QNAN =   0x7FFFFFFF;
const unsigned int SNAN =   0x7FC00000;
const unsigned int INF  =   0x7F800000;
const unsigned int ZERO =   0x00000000;
const unsigned int DENORM = 0x00000001;
const unsigned int NORM =   0x00800000;

//
//  This is a class definition for single precision IEEE-854 arithmetic.
//  Class real is by no means complete; it is the basic outline for
//  implementing a portable numerics system.
//

class real {
   private:
      union {
      float f;
      unsigned int i;
      };
   public:

      // constructors
      real();
      real(char *);
      real(float f);
      real(double d);
      real(char c);
      real(short s);
      real(int i);

      // No destructors really necessary

      // some basic primitives
      inline friend int isqnan(real &r);
      inline friend int issnan(real &r);
      inline friend int iszero(real &r);
      inline friend int isinf(real &r);
      inline friend int isdenorm(real &r);
      inline friend int isnorm(real &r);

      // overloaded operators
      friend real operator+( real &, real & );
      friend real operator-( real &, real & );
      friend real operator*( real &, real & );
      friend real &operator/( real &, real & );

      friend int operator==( real &, real & );
      friend int operator!=( real &, real & );
      friend int operator> ( real &, real & );
      friend int operator>=( real &, real & );
      friend int operator<=( real &, real & );
      friend int operator< ( real &, real & );

      inline void print(void) { cout << f; }

      }; // end of class real

// I/O related overloads

ostream &operator << (ostream &out, real &r) {
   r.printi();
   if (isqnan(r)) cout << " QNAN ";
   else if (issnan(r)) cout << " SNAN ";
   else if (isinf(r)) cout << " INF ";
   else if (isdenorm(r)) cout << " DENORM ";
   else r.print();
   return out;
   }
istream &operator >> (istream &in, real &r) {
   return in;
   }
real::real() {}  // allow uninitialized variables

real::real(char *str){
   if (strcmp(str,"INF") == 0)
      i = INF;
   else if (strcmp(str,"QNAN") == 0)
      i = QNAN;
   else if (strcmp(str,"SNAN") == 0)
      i = SNAN;
   else if (strcmp(str,"ZERO") == 0)
      i = ZERO;
   else if (strcmp(str,"DENORM") == 0)
      i = DENORM;
   else if (strcmp(str,"NORM") == 0)
      i = NORM;
   else {
      cout << "Unrecognized string format :" << str << ":\n";
      exit(1);
      }
   }

real::real(float fl){ f = fl; }

real::real(double d){ f = float(d); }

real::real(char c){ f = float(c); }

real::real(short s){ f = float(s); }

real::real(int I){ f = float(I); }

int isqnan( real &r ){
   unsigned int i = r.i;
   if ((i >= 0x7FC00000) && (i <= 0x7FFFFFFF))
      return 1;
   return 0;
   }

int issnan( real &r ){
   unsigned int i = r.i;
   if ((i >= 0x7F800001) && (i <= 0x7FBFFFFF))
      return 1;
   return 0;
   }

int iszero( real &r ){
   unsigned int i = r.i;
   if (i == 0)
      return 1;
   return 0;
   }

int isinf( real &r ){
   unsigned int i = r.i;
   if (i == 0x7F800000)
      return 1;
   return 0;
   }

int isdenorm( real &r ){
   unsigned int i = r.i;
   if ((i >= 0x00000001) && (i <= 0x007FFFFF))
      return 1;
   return 0;
   }
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ple that shows how this thinness can be a deficiency: what kind of exception should dividing a signaling NaN by zero generate? IEEE-854 does not specify. Signaling an invalid-operation makes sense because NaNs propagate, but signaling a zero-divide does not make sense because a NaN should take precedence. The result is thus a currently signed quiet NaN, not a currently signed (INF).

Intel's 80387 defines an exception priority list that gives more granularity than IEEE-854. The exception precedence is as follows [4]:

(1) Invalid operation exception, subdivided as follows: stack underflow, stack overflow, operand of unsupported format, and signaling NaN operand.
(2) Quiet NaN (QNaN) operand. Although this is not an exception, if one operand is a QNaN, dealing with it has precedence over lower-priority exceptions. For example, a QNaN divided by zero results in a QNaN, not a zero-divide exception.
(3) Any other invalid-operation exception or zero-divide.
(4) Denormal operand. If masked, instruction execution continues, and a lower-priority exception can occur as well.
(5) Numeric overflow and underflow. Inexact result (precision) can be flagged as well.
(6) Inexact result (precision).

Intel's i860 uses IEEE-854's exception precedence and does not define its own priority list. This is because the i860 floating-point unit is not stack oriented, does not do denormal arithmetic, and does not have a divide instruction. Thus, this priority list is not defined.

THE UNIQUE APPROACH OF C++

With its ability to create class definitions and overloaded operators, C++ provides a unique solution to implementing IEEE-854 arithmetic in a modern programming language. Consider overloading the predefined operators for *floats* and *doubles*. By creating class definitions to replace *floats* and *doubles*, the programmer controls the arithmetic being performed, including IEEE-854 exception handling. Operators can be created to treat anomalies proactively instead of reactively, as is the current trend. Consider writing a floating-point divide operator "/" that checks for division by zero, division of NaNs, and invalid operators before the operation occurs. Regardless of the underlying hardware architecture, these anomalies can be handled completely in application code, creating a high degree of portability and predictability. Each operation is captured by C++, and a function call is generated to the appropriate arithmetic routine. In this function, any type of range checking can be performed.

The trade-off for operand checking and portable IEEE-854 arithmetic is speed. Function calls and incessant checking of operand range slow simple operations to pre-8086 speed. Consider the C++ divide routine shown in Listing 1.

In the divide routine, both the numerator and denominator are checked for correctness before the division is performed. Although this creates a high degree of portability, it also creates a very slow divide routine.

Some of the attractiveness of reactive exception handlers is that the code needed to produce correct results is executed only when the results are guaranteed to be wrong. C++'s operator overloading and new exception-handling capabilities (throw, catch, etc.) allow for only proactive exception handling.

HEXADECIMAL NUMBER CONVERSION

Given the single-precision floating-point hexadecimal number "4CEB79A3," determine which single-precision floating-point number it represents.

- In binary notation:

  0100 1100 1110 1011 0111 1001 1010 0011
  4 C E B 7 9 A 3

  Sign Exponent Significand
  0 10011001 11010110111100110100011

- Determine the biased exponent:
  $10011001 = 2^7 + 2^4 + 2^3 + 2^0 = 128 + 16 + 8 + 1 = 153$
  Unbiased exponent = 153 – 128 = 25

- Determine the significand:
  11010110111100110100011 is not normalized.
  1.11010110111100110100011 is normalized;
  the 1. is hidden.

- Finalize the binary form:
  Shift the significand to the right by the unbiased exponent 25. Fill in zeros to the right where necessary:
  11101011011110011010001100000

- Convert back to hex:
  1 1101 0110 1111 0011 0100 0110 0000
  1 D 6 F 3 4 6 0

- Convert back to decimal:
  0x1D6F3460 = 493827168.0

Consider the following class definitions for *real*, a replacement for *floats* in C++. The basic operators (+, =, -, and /) are overloaded, as are the comparison operations (==, !=, >, <=, >=, and <). Seven constructors are provided. The most interesting of these allows *reals* to be initialized using strings:

```
real inf = "INF";
real qnan = "QNAN";
```

These initializations provide for the correct representations of infinities and NaNs. Additionally, the ostream "<<" and istream ">>" operators are overloaded for *reals* to ensure that anomalies are printed correctly (that is, infinities become "inf," Quiet NaNs become "qnan," signaling NaNs "snan," etc.).

In the C++ example in Listing 2, the divide operator checks both the numerator and the denominator for anomalies. NaNs get propagated correctly, divides-by-zero produce infinities (not core dumps), and denormals are detected but not manipulated. One important aspect of this divide operator is that the details that are usually hidden in an exception handler are all under programmer control. A programmer can easily adapt this exception handling to a specific application.

This simple program shows the expressive power of C++. A *real* can be defined using integral or floating-point constants or character strings. The output mechanism, *cout*, can be overloaded to print anomalies that can be read by users. Also, compares can be implemented to allow for anomalies, yet the program maintains the original spirit of C terseness.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
int isnorm( real &r ){
   unsigned int i = r.i;
   if ((i >= 0x00800000) && (i <= 0x7F7FFFFF))
      return 1;
   return 0;
   }

int operator == ( real &num, real &dem ){
   // this is a skeleton == operator for reals
   if (isnorm(num) && isnorm(num)) return (num.i == dem.i);
   else if (isqnan(num) && isqnan(dem)) return 1;
   else if (issnan(num) && issnan(dem)) return 1;
   else if (isinf(num) && isinf(dem)) return 1;
   else if (isdenorm(num) && isdenorm(dem)) return 1;
   return 0;
   }

const char DIV_BY_NAN = 0;
const char DIV_BY_QNAN = 1;
const char DIV_BY_ZERO = 2;
const char INVALID_OP = 3;
const char DIV_BY_INF = 4;

void set_psr(char c){
   // a stub for setting a user controlled processor control word
   }

real result;

real& operator/(real &nom, real &dem) {
   if (isqnan(nom) || isqnan(dem) || issnan(nom) || issnan(dem)){
      set_psr(DIV_BY_NAN);
      result.i = QNAN;
      return result;
      }
   else if (iszero(nom) && iszero(dem)){
      set_psr( DIV_BY_QNAN );
      result.i = QNAN;
      return result;
      }
   else if (iszero(dem)) {
      set_psr( DIV_BY_ZERO );
      result.i = INF;
      return result;
      }
   else if (isinf(nom) && isinf(dem)) {
      set_psr( INVALID_OP );
      result.i = QNAN;
      return result;
      }
   else if (isinf(nom) && isnorm(dem)) {
      set_psr( DIV_BY_INF );
      result.i = INF;
      return result;
      }
   else if ((isnorm(nom) || iszero(nom)) && isinf(dem)) {
      result.i = ZERO;
      return result;
      }
   result.f = float(nom.f) / float(dem.f);
   return result;
   } // end of real::operator/

/* A very simple program showing real assignments, divide operations,
**  output routines, and comparisons. */

void main(void) {
```

continued

People are talking about us.

F77L-EM/32 & Lahey Ergo OS/386

Port mainframe programs as large as 96MB to 386/486's with the 32-bit DOS-Extender compiler. The Winner of *PC Magazine's* 1989 Technical Excellence Award just got better. New Version 4.0 includes: Profiling Tools, Popular Fortran 90 features, Virtual Memory Support, DESQview Support, New Documentation and Free Unlimited Runtime Licenses. F77L-EM/32 $995 OS/386 $395

F77L

The fastest real-mode compiler available. F77L can take advantage of your 386 PC by generating 32-bit instructions. New Version 5.0 includes: Fortran 90 features, Weitek support, and Video Graphics. $595

Lahey Personal Fortran 77

Version 3.0: Full ANSI 77, Editor, Debugger, Linker, Library Manager, Microsoft and Borland C interfaces. A great learning tool at an unbeatable price. $99

Contact us to discuss our products and your needs. **(800) 548-4778**
Lahey Computer Systems, Inc. P.O. Box 6091, Incline Village, NV 89450
Tel: (702) 831-2500 FAX: (702) 831-8123

FORTRAN IS OUR FORTE

Program FASTER with ...

CC-RIDER

or the new

C++ RIDER

POP-UP SOURCE BROWSERS FOR ANY EDITOR!

CROSS-REF Editing!
Instantly pop-up your symbol definitions with a hypertext hot-key inside any editor! The Standard Edition has Source View, Edit and Paste commands. And the powerful *Professional Edition* even lets you walk through all uses of a symbol in your code as you edit!

Source Analysis!
Build a database of symbol information with our powerful ***source code analyzer*** which generates a wealth of useful output, like fully commented static and global function prototype #include files and a detailed log file showing WHERE and HOW all your symbols are used. And it uses expanded memory for ***lightning fast*** source code analysis!

QuickHelp, too!
CCSYM can also create a ***help database for your program*** which is compatible with Microsoft's help system, *QuickHelp*. All symbols in your application are accessible from QuickHelp's menus, with ***documentation extracted automatically from your original source code.***

Works with ANSI/Microsoft C, plus AT&T, Glockenspiel, Zortech, Borland C++. Supports Microsoft C v6 extensions.

FREE DEMO !

Standard	*Professional*
$89	***$279***
DOS or OS/2 C or C++	DOS & OS/2 C & C++

Western Wares
(303) 327-4898
Box C Norwood, CO 81423

Circle **35** on Reader Service Card

Circle **36** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
real r1;
real r2 = 0.0;
real r3 = 1.0;
real rinf = "INF";
real rqnan = "QNAN";
real rsnan = "SNAN";
real rzero = "ZERO";
real rdenorm = "DENORM";

cout << "r1 = " <<. r1 << "\n";
cout << "r2 = " << r2 << "\n";
cout << "r3 = " << r3 << "\n";
cout << "rinf = " << rinf << "\n";
cout << "rqnan = " << rqnan << "\n";
cout << "rsnan = " << rsnan << "\n";
cout << "rzero = " << rzero << "\n";
cout << "rdenorm = " << rdenorm << "\n";

r1 = r3 / r2;   // this should generate an INF
cout << "r1 = INF ?? " << r1 << "\n";

r1 = rqnan / rqnan;  // this should generate a QNAN
cout << "rqnan = QNAN ?? " << rqnan << "\n";

if (r1 == rsnan)
   cout << "WRONG!!! QNAN == SNAN\n";
else
   cout << "RIGHT!!! QNAN != SNAN\n";

} // end of main
```

LANGUAGE SOLUTIONS

Implementing IEEE-854 arithmetic and its associated exception handler takes an intimate understanding of the processor's architecture. Solutions offered by compiler vendors for providing a conforming environment vary greatly between languages and target processors. A conforming IEEE-854 implementation is usually made up of pieces from the processor, runtime libraries, compiler-generated code, and perhaps even language constructs, as we examined in C++. Additionally, a language implementation such as C++ could go so far as replacing all numeric types (*floats* and *doubles*) with an arithmetic package that performs all floating-point calculations using integer operations, thus realizing IEEE-854 mostly in software.

Because IEEE-854 is not well understood (or well implemented) by many compiler vendors, it is important for competent compiler users to understand the issues in providing correct floating-point results.

REFERENCES

1. J. Coonen et al., "A Proposed Standard for Binary Floating Point Arithmetic," *ACM SIGNUM Newsletter*, Oct. 1979.
2. Institute for Electronics Engineers, *IEEE Standard for Radix-Independent Floating-Point Arithmetic*, 1987.
3. Intel Corp., *i860 Microprocessor Programmer's Reference Manual*, 1990.
4. Intel Corp., *80387 Programmer's Reference Manual*, 1987.

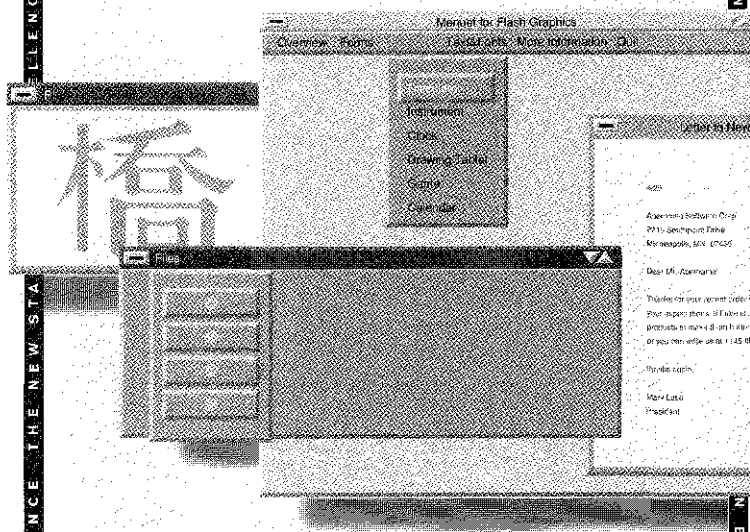
Additional reading:

- Intel Corp., *i860 Microprocessor Hardware Reference Manual* and *iAPX User's Manual*.
- Neal Margulis, *i860 Microprocessor Architecture* (Berkeley: Osborne McGraw-Hill, 1990).
- MicroWay, Inc., *NDP-486-FORTRAN Manual*, 1990.

THE NEW STANDARD IN EXCELLENCE THE NEW STANDARD IN EXCELLENCE

Menuet™

The New Standard in Excellence in GUI Design for DOS, 386 Protected Mode, and UNIX!



Versions for MetaWINDOW, GSS*CGI, SCO*CGI, and Zortech Flash Graphics

Includes Interface Design Tool, Menu Design Tool, a Font Editor, and an Icon Editor

Over 1,000 pages of printed documentation in 2 volumes, plus full on-line documentation

Easy to use - we've had customers create demos in 3 days, finished applications in 3 weeks

Simply outstanding aesthetics - Emulate WINDOWS, MOTIF, or your own unique look!

DOS Version: **$325** UNIX Version: **$650**

Call for your free demo!

303.494.8865

AUTUMN HILL SOFTWARE

Circle **37** on Reader Service Card

1145 ITHACA DRIVE, BOULDER, CO, 80303, FAX: (303) 494-7802 BBS: (303) 494-8868

FORMERLY ITHACA STREET SOFTWARE. ALL TRADEMARKS ARE THE PROPERTIES OF THEIR RESPECTIVE OWNERS.

THE NEW STANDARD IN EXCELLENCE THE NEW STANDARD IN EXCELLENCE

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C CODE FOR THE PC

source code, of course

NEW!	Drover's ToolBox for Windows (200 functions in one DLL, 7 new dialog classes)	$750
	M++ (fast C++ math classes; indexing, matrices & vectors, numerics, memory handling, I/O; specify C++ compiler/Turbo, Zortech, Glock.)	$395
Windows!	C++Views (C++ interface to MS Windows 3.0; over 60 classes; free broswer (no source); no royalties; Zortech C++ only)	$395
NEW!	CAD/CAM Developer's Kit (large collection of math, graphics, display & DXF subroutines for CAD/CAM/CAE/CNC; math coprocessor req'd)	$375
	MBALLOC (best-fit far heap manager, replaces standard routines, Microsoft performance on Turbo C)	$340
	ZIP Image Processing Library (brightness, contrast, merge images, TIFF/GIF/PCX/bin, printer & HRT 512 digitizer support)	$290
	Victor Image Processing Library (EGA/VGA; TIFF/PCX/GIF/bin; multiple images; many filters; frame grabber & live video)	$250
NEW!	MemCheck Memory Debugger (excellent library to detect buffer overwrites, unfreed pointers, etc., by source file and line)	$250
	TurboTEX (Release 3.0; HP, PS, dot drivers; CM fonts; LaTEX; MetaFont)	$250
	db_File & db_Retrieve Bundle by Raima (B-tree and network database with SQL query and report writer; multi-user $475)	$245
	Install 3.05 (automatic installation program; user-selected partial installation; CRC checking)	$195
NEW!	TE Editor Developer's Kit for Windows (full screen editor, undo command, multiple windows; inc. TER for application build-in; no royalties)	$220
NEW!	String Externalization Tools (one .exe, many languages; change language of prompts & messages by changing external file; no royalties)	$210
NEW!	Report Right (report writer and graph generator for Windows 3.0)	$170
NEW!	C-Index/PC Database Library (fast B+ tree indexing, string/binary/custom keys, multiple record formats/file, 170-page manual)	$165
	Rogue Wave tools.h++ or math.h++ Class Library (extensive docs)	each $165
NEW!	DXF Subroutine Libraries (toolkit for making AutoCAD DXF drawing files; includes manual & sample programs)	$155
	WKS Library Version 2.01 (C program interface to Lotus 1-2-3, dBase, Supercalc 4, Quatro, & Clipper)	$155
Windows!	RED (program development editor for Microsoft Windows; keystroke recording & playback; unlimited undo)	$150
	Minix Operating System (Version 1.5; Unix-like operating system, includes manual; specify 5.25" or 3.5" diskettes)	each $150
NEW!	Delorie GCC for MS-DOS (Version 1.39; includes C++, assembler & DOS extender; complete source code and makefiles)	$150
	C Generators (specify C, C++ or Windows)	$140
	Vmem/C (virtual memory manager; least-recently used pager; dynamic expansion of swap file)	$140
	CBTree (B+tree ISAM driver, multiple variable-length keys)	$135
	TE Editor Developer's Kit (full screen editor, undo command, multiple windows)	$130
	C Communications Toolkit by Magna Carta (multi-port & co-processor board support, FAX, interrupt driven, emulations, xfer protocols)	$125
	C++ Object Library (virtual windows, I/O, lists, file free space management, keyed ISAM file I/O)	$125
	SilverWare "C" EMM Library (60 interface functions to EMM Version 4.0)	$120
Windows!	WinMem (unlimited global memory handles for Windows 3.0, 4-byte overheadper block rather than 20, virtual memory for 286)	$110
	Network Toolbox/N (C functions for Novell Netware 2.1x, sample programs included)	$110
	PC/IP (CMU/MIT TCP/IP for PCs; Ethernet, Appletalk & NETBIOS drivers, RVD, Clarkson drivers, Stan's NFS clone, Bdale mailer)	$100
	PowerSTOR (Version 1.2; extended heap space on extended memory, expanded memory, and/or hard disk)	$95
NEW!	MIRACL Multiprecision Integer & Rational Arithmetic C Library (nearly 100 functions; C & C++; from Ireland)	$90
	Disassembler (8086/80286/80386/80C186 with 80287/80387/80C187 NPX)	$90
	Booter Toolkit (floppy disk bootstrap routines, DOS file system, light-weight multitasking, windows, fast memory management)	$85
NEW!	VM4C (virtual memory manager in C; swaps blocks to disk; small, fast, simple)	$80
Upgrade!	Virtual Memory Manager by Blake McBride (LRU pager, dynamic swap file, image save/restore)	$80
	MultiDOS Plus (DOS-based multitasking, intertask messaging, semaphores)	$80
NEW!	BGI Printer Driver Toolkit (BGI drivers for Epson 9- & 24-pin, HP LaserJet II, & PaintJet printers)	$75
	COMM-DRV (interrupt-driven serial communication, device drivers & serial port monitors; source for libraries only; complete source $200)	$75
	XSPAWN (swap parent process to expanded memory or disk before running child)	$70
Upgrade!	FlexList (doubly-linked lists of arbitrary data with multiple access methods; specify C or C++)	each $65
	Foundations-1 C++ Class Library (ASCII record I/O, bit arrays, exception handling, B-tree, persistent objects)	$60
	MultiStr (evaluates a string containing multiple ASCII arithmetic expressions with variables)	$60
	Coder's Prolog (Version 3.0; inference engine for use with C programs)	$60
NEW!	FinanC (large collection of financial function including bond, inventory, stock portfolio, & cash flow)	$55
	Pascal P-Code Compiler & Interpreter or Pascal-to-C Translator (Level 0 ISO standard Pascal with some Level 1 features)	each $50
	CALC (ASCII algebraic expression evaluator, unlimited parenthesis nesting, symbols, 32 built-in functions, easily extended)	$50
	Backup & Restore Utility by Blake McBride (multiple volumes, file compression & encryption)	$50
	Floppy TAR (TAR backup and restore on MS-DOS devices; direct access to non-standard devices)	$50
	SuperGrep (exceptionally fast, revolutionary text searching algorithm; also searches sub-directories)	$50
	OBJASM (convert .obj files to .asm files; output is MASM compatible)	$50
	YATE (Yet Another Text Editor; written in C++, full-screen)	$50
	CLIPS (rule-based expert system generator, Version 4.3; advanced manuals available at additional cost)	$50
	NIH Class Library & Book (basic C++ classes & *Data Abstraction and Object-Oriented Programming in C++* in softback by Keith Gorlen)	$50
	Kier DateLib (all kinds of date manipulation; translation, validation, formatting, & arithmetic)	$45
	DES Encryption & Decryption (2500 bits/second on 4.77 MHz PC for on-the-fly encryption at 2400 baud; domestic distribution only)	$40
Upgrade!	RXC & EGREP Version 2.0 (Regular Expression Compiler and Pattern Matching; finite state machine from regular expression)	$35
	Bison & BYACC (YACC workalike parser generators; documentation; no restrictions on use of BYACC output)	$35
	PC-XINU (Comer's XINU operating system for PC)	$35
	Spell Pack (6 spelling programs, a hyphenator, 2 utility packs and a 60K word list: Ispell, Microsp, Sp, Cspella, Spell, Dawg, Soundex)	$30
	REGX Plus (search and replace string manipulation routines based on regular expressions)	$30
	GNU Awk & Diff for PC (both programs in one package)	$30
Upgrade!	6-Pack of Editors (ten public domain editors for use, study & hacking; includes microEmacs 3.10, Stevie, Elvis & Origami)	$30
Upgrade!	Crunch Pack (25 file compression & expansion programs)	$30
NEW!	Make Pack (eight versions of Make including GNU Make, Cake, and Gymake and a makefile maker)	$30
NEW!	BigNum (portable and efficient arbitrary-precision arithmetic package; hardcopy docs; from France)	$25
	String Pack (lots of string routines; C++ String class, BAWK, word wrap, fuzzy search, Boyer-Moore, Hypertext, etc.)	$25
	PC-MAIL (UUCP mailer by Wietse Z. Venema; send, receive, and manage UUCP mail; from Holland)	$25
	PERL for MS-DOS (C, sed, awk, and shell all rolled into one language; includes hardcopy docs)	$25
	FLEX (fast lexical analyzer generator; new, improved LEX; BSD Version 2.3.6 with docs)	$25
	Using C++ Library (the code from the book by Bruce Eckel and then some; Zortech 2.0 compatible)	$25
NEW!	GNU Chess (GNU Version 3.1 & Windows Version 1.01; executable included; requires SDK to build; truly beautiful!)	$20
NEW!	GNU RCS (FSF's version of the Revision Control System; like Unix's SCCS only better; keeps track of software development)	$20
NEW!	SDBM (fast, disk-based hash table manager for really large hash tables; clone of Unix *ndbm*)	$20
NEW!	NLMDL (non-linear least squares fitting in C++)	$20
	Unix/386	
NEW!	Kyoto Common Lisp (Austin flavor, naturally; Version 1.530; includes GCC and Portable Common Loops (PCL))	$140
	GNU Emacs Version 18.55 (includes complete source & executables)	$125
	Waffle BBS (interface to UUCP & B or C news; internal forums; files section; external processing; full USENET support)	$120
	GNU C Compiler (gcc) Version 1.39 (includes complete source & executables)	$100
	ViewComp Spreadsheet (internal termcap; printer output; hardcopy docs)	$80
	GNU Debugger (gdb) Version 3.5 (includes complete source & executables)	$60
NEW!	Gillespie Pascal-to-C (many flavors of Pascal handled; readable & maintainable output)	$25
NEW!	Gosling Spreadsheet Pack (variants of the 1982 Gosling spreadsheet including PubliCalc and SC version 6.10; 1 MS-DOS variant)	$25

The Austin Code Works
11100 Leafwood Lane
Austin, Texas 78750-3587 USA

Voice: (512) 258-0785
FAX: (512) 258-1342
E-mail: info@acw.com

Free surface shipping for cash in advance **For delivery in Texas add 7%** **MasterCard/VISA**

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C

Virtual realities

Implementing an application-supported virtual memory manager

By Stephen M. Kuhn

Demands on applications are increasing rapidly. They are expected to do more—and do it more quickly—than ever before. Witness the evolution from simple character-based spreadsheets to those that incorporate presentation graphics, or from simple word processors that could run in less than 64K to full-blown page layout programs requiring 2MB to perform reasonably well.

Compounding this problem of proliferating demands are multitasking environments like Windows and DESQview that can operate two or more large applications concurrently. Fortunately, disk space and RAM are getting less and less expensive, so that a 100MB disk coupled with two, four, and even eight megabytes of RAM is becoming commonplace in nondevelopment computers.

VORACIOUS MEMORY

So where's the problem? DOS. Its 640K limit is peanuts.

Most of the world doesn't realize that there's a problem because environments such as Windows 3.0 and DESQview 386, along with DOS extenders, hide the 640K limit or circumvent it by switching the computer into protected mode. Expanded memory is out, extended memory is in—and with it comes a new generation of great software.

However, software developers are becoming aware that DOS is inadequate to handle the voracious memory requirement of today's sophisticated applications. So how can DOS developers create applications that use more than 640K?

The same way Windows does it. The same way Unix, OS/2, VMS, and many other operating systems do it—with virtual memory (VM).

A LOOK AT VIRTUAL MEMORY

Virtual memory maps logical address space into physical memory (RAM). Because the computer can directly operate only on data stored in RAM, information that is not currently being used must be stored elsewhere. When that information is needed to perform some operation, it must be brought back into RAM. The storage device is usually a disk, but in principle it could be any read/write device. RAM consequently acts as a sliding "window" into the entire address space.

There are two tricks to virtual memory: one is knowing what is not being used and can be stored on the disk (swapped out); the other is knowing when information not currently in RAM needs to be brought back (swapped in).

To have virtual memory that is transparent to applications, both the operating system and the hardware architecture must support it. (Actually, hardware support is not required, but it greatly facilitates the implementation and the resulting speed of virtual memory.) When an application uses a piece of memory in a virtual-memory machine, the CPU makes sure that the expected information actually is in the expected piece of memory. If it isn't, the CPU automatically brings it in from the disk. The application never knows—or needs to know—anything about the process. Furthermore, because the OS and the CPU are in control, applications themselves can be swapped in and out of memory.

VIRTUALIZING DATA

Computers running plain DOS (i.e., without Windows, DESQview, or other such programs) have no virtual memory; the early 8086 CPUs and the DOS operating system were not designed to support it. Therefore, applications needing to access more memory than is physically available must implement **application-supported virtual memory** (ASVM) on their own.

Different applications have different needs, so the required type of ASVM varies. For programs that have large data needs, several questions should be answered before you embark on a particular strategy.

What data requires virtualizing? If you have only one big piece of data, a simple scheme may work. However, a

Stephen Kuhn is co-founder and president of The Iliad Group, Inc., and the principal designer and engineer for their PI Edit. He can be reached at (415) 563-2053.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C++
Professionals!
Linnaeus
The C++ class browser and design tool
Instant access to detailed information about every class on your machine - just click on its icon to display all the in-depth information you need
Graphical trees let you examine complex inheritance heirarchies, whether the classes are in one file or *hundreds* on multiple drives
Ad-hoc query lets you find *any* class or *list of classes* quickly
Innovative GUI environment
$195
VMO
Dynamic Virtual Memory Objects
Compare. No one else gives you all of these features:
Btrees, strings, arrays, lists, m-trees, heaps, stacks, more.
Variable length element sizes to 4 gigabytes.
Uniform data access. Data I/O independent of physical location.
No more malloc. Object allocation, growth and swapping are done transparently as needed. Or not. You decide.
No more fopen, fscanf. Files are treated like memory. Use MemCpy, MemCmp, etc.
Persistent objects, of course.
No royalties, naturally.
Messaging. Built-in support for window-style messaging.
Full documention and support
Over 70 classes, 1000 methods
Only $295!
Zircel Software
(303) 750-9543
Circle 39 on Reader Service Card

large application, such as a spreadsheet, probably wants all of its data (cells, formulas, formatting, values, etc.) to be in virtual memory.

Does the data consist of fixed-length records? Implementing a virtual memory system is much easier if you have fixed-length records. Variable-length data poses a more difficult problem.

What is the range in size of the data you need? Does your application use small data variables or does it use huge structures and arrays? The size of data you use will dictate the overhead required to virtualize data.

Will data be accessed randomly or sequentially? For sequential data access, you can employ simple caching and read-ahead algorithms to optimize speed. However, optimizations for random access are harder to implement and require more sophistication and overhead.

Does new data need to be dynamically allocated and freed during execution? Dynamic allocation greatly increases the complexity of the ASVM. In the remainder of this article, I will use a DOS text application to examine some strategies for implementing an ASVM.

A BLOCK-ORIENTED APPROACH

One approach to data organization is to treat the file as one large buffer. The whole file is read from the disk and stored in memory in the same format it had on the disk; you essentially move the text from a disk file to a memory file.

Let us assume that this virtual-memory application starts by grabbing a chunk of RAM from the heap. We don't want to use the disk until the RAM buffer has been exhausted. When that occurs, the excess text must spill onto the disk. The RAM buffer then becomes a *single* large window through which we view and edit the text. As the user's focus in the file shifts, the window must move accordingly. This is analogous to the screen's acting as a window into the RAM buffer.

One variation on this method is to maintain the buffer in two parts: one before the focus (usually the cursor position), and one after. This method requires less shifting of the text within memory as text is inserted and deleted. Managing these two windows (essentially two buffers) is more complicated, but the resulting performance is usually better.

BENEFITS AND DRAWBACKS

Overall, this approach has some definite benefits (see Table 1). It requires no memory allocation and free functions and therefore eliminates the need for garbage-collection and defragmentation routines. The trickiest part is moving the RAM window(s) through the text and rolling the buffer contents onto the disk in a sensible fashion.

Another benefit is the very low data overhead that results from having all of the data in one or two buffers. In addition, disk I/O is faster because the text can be read and written with no translation or parsing into component structures (such as lines).

However, there are some strong drawbacks. Because many text operations may involve working with different parts of the file (like copying a block of text), having only one window to the file may require data to be swapped in and out more often, resulting in slower operation.

Although it requires little code devoted to the virtual-memory system, the block approach may require virtual-memory code to be integrated into the rest of the program. For example, the code to copy a block of text must have a built-in sense of how the text is organized in memory, even perhaps swapping data in and out to the disk.

TABLE 1: Block approach for text applications

Pros
Fast disk I/O
Easy modification of text
Simple data structures
Low data overhead
No memory allocate or free function requirements
Cons
Requires smart display routines to parse the text correctly
May be cumbersome when dealing with large blocks
Slower in determining column and line number and other information
Does not apply well to other applications of virtual memory
Only text can reside in virtual memory
Difficult and slow with large text operations
The code must "know" too much about the underlying data structures

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

A LINE-ORIENTED APPROACH

Text files are conveniently organized into small pieces—lines and arrays of lines or pointers to lines. Arrays are usually unacceptable because they are typically of a fixed size, and text files have neither a fixed number of lines nor a fixed line length. To edit large files, we need to implement virtual memory.

Having decided to break the file down into smaller pieces, let us revisit the questions posed earlier. For a text editor (or word processor):

- All of the data is virtual.
- The data is of many different sizes.
- The virtual data may be large—from one byte to many kilobytes.
- The data can profit from random access.
- The data may need dynamic allocation.

With these needs, it makes sense to use a generalized virtual-memory manager. This generalized VM manager would be able to set up a swap area, service memory allocation and free requests, perform garbage collection, and manage the use of that memory. However, there are several different ways to organize the memory and the data structures in a text application, and these will greatly influence the type of memory manager we implement.

It is important to consider how text typically is manipulated. Modifications are usually done on individual lines (e.g., typing) or on groups of text lines (copying or moving a block) rather than on blocks that begin and end on arbitrary boundaries. For program editors it is rare to have a block begin in the middle of a line, end in the middle of another line, and be copied to some other place; in word processors, this sequence is more common.

LINKED LISTS OF LINES

A line-oriented approach introduces entirely different conditions on a virtual-memory manager. Now there are lines that need to be ordered and linked. An attractive approach is a doubly linked list of lines. Suddenly each line of the file has instant overhead, but also more information is available than with the block-oriented scheme. The line structures can also be expanded to hold additional information such as video attributes, block attributes, and line numbers.

We no longer have big pieces of data (a few blocks containing the entire file), as was the case under the block scheme. We now have many more items, including an unknown number of data items. It makes sense to allocate the data structures and line buffers from a VM heap.

Each allocated item can no longer refer to a specific location in memory, for the simple reason that it may not stay there—the VM manager may need to put it on the disk in order to bring another needed bit of data in or to accommodate a VM allocation request. While **malloc** might return an address like 49f5:0005h on the PC, the VM allocator returns a *token* or *handle* (for example, 0ffa9eh, which is an invalid memory address on an MS-DOS PC). This token is unique for every allocated item. Each reference to the handle causes the VM manager to ensure that the data is in RAM by loading it from the disk and swapping something else out to disk if necessary. This is in fact what happens in virtual operating systems.

PC-lint 386 Computer Language Productivity Award 1990

PC-lint presents C Bug # 574

```
#include <stdio.h>
main()
    {
    unsigned row = 25, col = 80;
    if ( row + col < -1 ) printf ( " error " );
    }
```

What output does your C compiler produce and what warnings do you get? Is the output what you expected? If you can't find this bug, give us a call.

PC-lint will catch this and many other C bugs. Unlike your compiler, PC-lint **looks across all modules** of your application for bugs and inconsistencies.

"PC-lint, the one tool I believe *all* C programmers should have."
The C Gazette

More than **270 error messages.** More than **90 options** for complete customization. Suppress error messages, locally or globally, by symbol name, by message number, by filename, etc.
Easy to Use – If you know C you already know how to use PC-lint. Check for **portability** problems. Alter size of scalars. Adjust format of error messages. Automatically generate ANSI prototypes for your K&R functions.

New Features in Version 4.0
Lint Object Modules to support incremental linting
Weak Definial Checking – macros, structs, typedef's, enums, and declarations not used globally and locally. Header files not used. Externals that can be made static. Declarations that can be moved from headers.

Mainframe & Mini Programmers
FlexeLint in shrouded source form, is available for Unix, OS-9, VAX/VMS, QNX, IBM VM/MVS, etc. Requires only K&R C to compile but supports ANSI. Pricing starts at $798. Call for details.

Gimpel Software

3207 Hogarth Lane, Collegeville, PA 19426
CALL TODAY (215) 584-4261 Or FAX (215) 584-4266

PC-lint is only $139 – Specify MS-DOS or OS/2
PC-lint 386 is only $239 – Requires MS-DOS on a 386 *(For power users with huge programs)*
30 Day Money-back Guarantee.

PA add 6% sales tax.

PC-lint and FlexeLint are trademarks of Gimpel Software

Contact this advertiser directly.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

FIGURE 1: Application-supported vitural memory

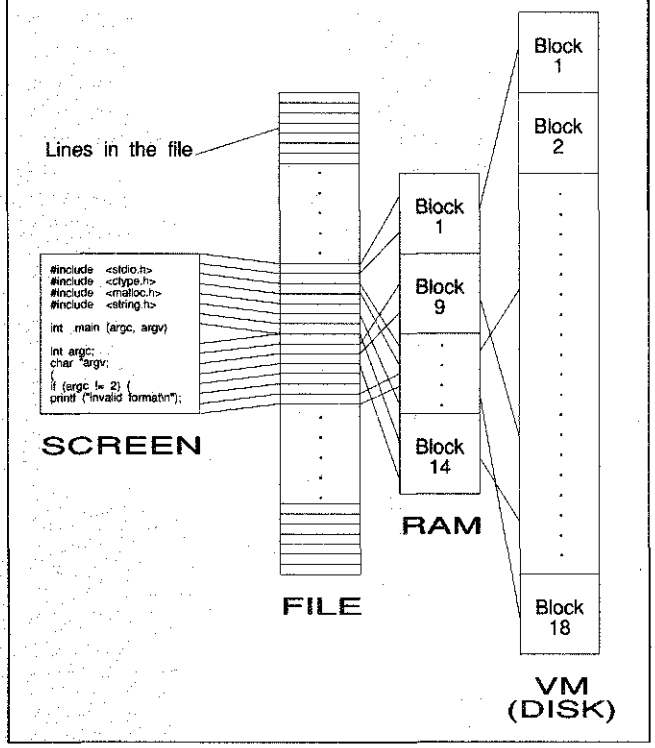
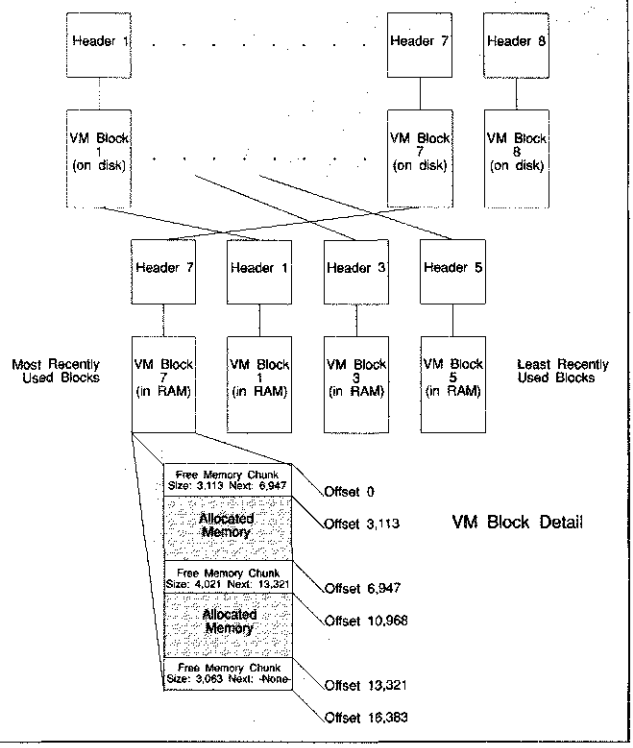


FIGURE 2: VM manager structure



HERE'S THE GLITCH

Of course, this method contains a serious problem: tokens are not RAM pointers and cannot be used as such. Consider the following C code:

```
char far *p;

p = (char far *)0xffa9e;  /*from example above*/
strcpy(p, "This will not work!");
```

The variable *p* on MS-DOS systems will definitely *not* point to the text "This will not work"; in fact, on Unix systems this code will almost invariably result in a segmentation violation (or worse). The processor is not monitoring memory references and then loading the required data into RAM. So how can we use tokens instead of real pointers and have them really work? Actually, the answer is simple—every time we need a piece of allocated data, we must explicitly ask to use it.

Consider the following C fragment:

```
typedef   unsigned long TOKEN;
TOKEN     vmalloc(int);
char      *vmderef(TOKEN);
TOKEN     t;
char      *p;
if ((t = vmalloc(1000)) == (TOKEN)0)
   return(FAILURE);
strcpy(vmderef(t), "This WILL work");
```

The function **vmalloc** is the virtual-memory allocator, and it returns a token uniquely referring to the new 1000-byte block of memory. The function **vmderef** calls the VM manager, which ensures that the 1000-byte array is loaded into RAM for use. This looks easy, and it is. The underlying code, however, is a bit more involved.

PICK A STRUCTURE

First we need to decide on a structure for the ASVM in our text application. We want to use the disk as our swap device, and let's assume that we support EMS memory as well (the standard way to extend DOS RAM). In choosing a structure we must balance speed, use of RAM, and simplicity of implementation.

When we dereference a token (that is, swap it in from the disk), ideally we want to swap in only that item. This implies that each VM item is independent from every other item. One may be in RAM and another may not, and bringing one into RAM should not affect another. Unfortunately, maintaining VM items in this way can require a huge overhead to know the exact location of each item (both in RAM and on the disk). Implementing the memory management for such a system is burdensome and slow.

We can group VM items into blocks. The question is, how large should we make these blocks? If they are too small, we limit the size of allocated VM items. If they are too large, the VM manager is limited in the number of items it can have in RAM at any one time.

EMS memory uses four 16K RAM "windows" to access expanded memory. We can allocate 16K blocks from RAM, and the size is perfect to support EMS memory. By doing so, however, we are limiting the maximum size of an item allocated with the VM manager to 16K. Therefore, no line or other data item can exceed 16,384 bytes. This limit is reasonable for most text applications.

The tokens must describe the location in VM of the item to which they refer. We must keep two pieces of information to track the location of each VM item: which VM block it is in and its position within that block.

Because we are using 16K blocks, we need 14 bits (2^{14} = 16K) to store an item's position in the block. If we use a standard 16-bit value (the size of a short integer on PCs), we have only two bits on which to store the block number, and we will certainly need more than 64K ($2^2 \times 16{,}384$) of virtual-memory space.

Therefore, we will use a 32-bit value

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

(an unsigned long integer on PC machines). The first 16 bits (one word) will contain the block number, and the second 16 bits (only 14 of which will actually be used) will store the item's position in the VM block. This allows us to address a gigabyte (64K × 16K) of data. If we assign the 2 unused bits to the block number, then the size increases to 4 gigabytes.

WORTH A THOUSAND WORDS

If we *really* need more data, we could increase the token size to 48 or 64 bits and use 32 bits for the block number. However, long-integer math is slower. Furthermore, how often does one need to edit thousands of gigabytes of text?

Figure 1 (p. 54) diagrams this memory scheme. Lines of the file are stored in virtual memory in the VM swap file. Lines that are needed by the application for modification or reference (for example, to display them on the screen) are stored in VM RAM blocks. Because of editing changes such as moving a line from one location in the file to another, file lines do not necessarily reside in adjacent VM blocks.

The RAM blocks can be filled with any blocks from the VM disk file—and in any order. They are swapped in and out as required. The RAM and VM disk blocks may also contain other lines from the file, line structures from the file, and any other data allocated through the VM manager.

There are four main component subsystems in the virtual-memory system: the initializer, the allocator, the swapper, and the terminator. Each has a very important task in a VM implementation.

The **initializer** sets up the VM manager. It first determines the amount of available RAM and the status and availability of EMS memory and then allocates RAM buffers, creates the VM swap file, and initializes both.

Because the only RAM used in the application is within the VM manager's RAM buffers, the **allocator** must implement a complete memory management mechanism, including both **allocate** and **free** routines. **Free** must also have a means by which contiguous available chunks of memory are reconnected to form a single, larger block of available memory.

As tokens are dereferenced, different VM blocks from the disk will need to be swapped in, taking the place of VM blocks currently residing in RAM. The **swapper** handles this job. Unlike the allocator, it does not modify the contents of VM blocks in any way; it simply manages their movement and location.

When the VM manager is no longer needed (such as when the application terminates), the **terminator** closes and deletes the VM swap file and frees the VM RAM buffers.

EXPANDING THE MODEL

We have now developed a model for a text-based, application-supported virtual-memory management system. The data used by the application is organized in RAM buffers of 16K. These buffers may be in conventional or in EMS memory (or in extended memory, if we implement support for it). As data is needed, the corresponding VM block is swapped in from the disk and another is swapped out.

Figure 2 shows a possible structure of the VM manager. VM blocks are described by a VM header and are allocated on an as-needed basis. They reside on the disk or in RAM (either conventional or EMS). Those in RAM have been swapped in to be used or have just been created and have not yet been put onto the disk.

Blocks currently in RAM are chained together in a doubly linked list. The VM blocks at the head of the list are those that have been most recently accessed; those at the end of the list have not been. This method of linking allows us to implement a most-recently-used (MRU) algorithm.

Using an MRU (or an LRU—a least-recently-used) swapper ensures that the VM blocks that have been used most recently are the last ones to be swapped out when RAM is needed. This procedure increases the performance of the VM manager by preventing the swapper from constantly swapping in and out the same (needed) blocks.

Chunks of allocated and deallocated (available) memory are within each VM block. The allocator is responsible for maintaining these blocks, and each block has its own list of free memory chunks. The free chain is sorted in ascending order: that is, any item in the free chain is lower in memory than the next free chunk. Initially, a VM block has one free chunk, which is the size of the block.

WE'VE CHARTED THE WAY WITH WIN++

...for Windows applications in Borland C++

We've chosen the direction and plotted the course to easy and powerful Windows applications development!

With nearly 100 classes, Win++™ provides a simplified, flexible and powerful high-level C++ interface making your Windows development efforts a breeze.

Put the wind at your back...

Central to Win++ is that it provides a class for each type of display object in Windows. Win++ includes ready-made classes for representing dialog boxes, child controls, pens, brushes, menus, bitmaps, the cursor, icons, a printer, the clipboard and many others.

Win++ also supports the construction of dynamic-link libraries (DLLs). Use Win++'s ready-made DLLs in your programs and save valuable time, memory and disk space.

It's smooth sailing from here out...

Win++ comes with a comprehensive reference manual and tutorial. Complete source code is included. You can rely on Blaise Computing to provide and support quality add-on libraries. We've been doing so since 1983 — it's our business — our only business.

Success is in the offing...

Win++ requires Windows 3.0 and Borland C++ Professional version 2.0 or later. Win++ costs just $249. We're so convinced that you'll find Win++ essential to your applications that if during the first 60 days you're not completely satisfied, we'll refund your money. That's a 60-day money back guarantee!

Call our order department toll free at **(800) 333-8087**, today! Or, by fax, (415) 540-1938.

BLAISE COMPUTING INC.

819 Bancroft Way Berkeley, California 94710 (415) 540-5441

Trademarks are property of their respective holders.

Circle 40 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

BLOCK MAINTENANCE

The initializer determines whether there is enough RAM to allocate the VM RAM blocks (in this case, 16K each). If there is, it allocates them. It also initializes the VM headers that describe each VM block.

In addition, the VM manager needs to maintain several pieces of information for each VM block. It is most convenient to organize them into data structures that refer to a single VM block. Listing 1 outlines the **vmheader** data structure for each VM block.

Unfortunately, we do not know the number of VM blocks that we will need ahead of time: we may need many, we may need few.

One way to handle this is to allocate an array of arbitrary size (for example, 256K) of these structures using **malloc**; if the VM manager needs more blocks, it can reallocate the array to a larger size. There is always the chance, however, that **malloc** will be unable to find enough memory to allocate the larger array, forcing the VM manager to return an artificial out-of-memory condition, which is not desirable.

A flexible method is to keep the VM block information on the disk in a separate disk file and swap it in and out as needed in its own mini-VM system. This is easy because the VM blocks are of a fixed size and are not deallocated. We just need to index into a disk file with fixed-length records and read or write the blocks. Maintaining the blocks in a disk file is, however, slower than the RAM array method.

THE SWAPPER

It is important that the VM manager be as fast as possible. Most of its speed can come from doing just a few simple things. One of these is to use the MRU scheme described above. Another is to know, when a block is to be swapped out to the disk, whether the block has been changed and *needs* to be written; if it hasn't been changed, then not writing it out saves time. Because the VM manager does not know whether the caller will merely reference the memory block or actually update it, the practical way to let it know is for the caller to tell the swapper. The function **vmderef** would then take a second argument of *0* if the caller is only going to look at the RAM data, or *1* if the data is to be modified. If the data will be modified, the swapper changes the *dirty* field in the **vmheader** structure for the VM block to *1*.

When the VM manager function **vmderef** is called, it splits the passed token into the VM block number and the offset of the memory chunk within that VM block. It then determines whether the VM block is already loaded into RAM. If it is not, the function swaps out the VM block at the end of the MRU chain. First, it writes the block to the disk (if the block has been modified), updates the *location* field in the **vmheader** structure for that block, and removes the **vmheader** from

SAY GOODBYE TO "OUT OF MEMORY" ERRORS!

You bought your 386 for some very good reasons. Power. Speed. Performance. But are you really getting the most out of your investment? Without memory management software, chances are the answer is no. 386MAX and BlueMAX let you get MAXimum performance out of your 386. Our memory managers give your software the room it needs to run faster and more efficiently.

Run Your 386 to the MAX!

DISCOVER QUALITAS TECHNOLOGY!

Qualitas memory management products deliver quality, reliability, and innovative optimization features you won't find anywhere else:

- **FlexFrame**—The newest innovation in memory management technology. FlexFrame loads more programs into high DOS than ever before by temporarily removing the EMS page frame during TSR initialization.
- **Guaranteed EMS Page Frame in high DOS**—Using BlueMAX, the EMS page frame is now located efficiently and easily in high DOS on PS/2 systems.
- **Automatic Adapter Relocation**— Only 386MAX and BlueMAX automatically readdress Micro Channel adapters to give you larger, contiguous blocks of high DOS memory.
- **Complete Diagnostic Tools**—Both 386MAX and BlueMAX come equipped with an exhaustive set of diagnostics that let you pinpoint memory usage.

PS/2 CONTROL!

BlueMAX is the newest addition to the Qualitas family of memory management products. When BlueMAX manages your system's memory, you recover an additional 80+ KB high DOS memory! BlueMAX's unique technology provides the ideal memory management solution for 386 PS/2 users.

NOW RUN WINDOWS TO THE MAX!

Windows 3.0 user? Our memory managers improve Windows performance by moving resident software out of low DOS memory, giving Windows more room to run. 386MAX and BlueMAX are the *only* memory managers that provide *instancing* capabilities for TSRs. This exclusive feature allows many TSRs to run completely independently in every DOS window you open!

386MAX® BlueMAX™

386MAX: $130.00
Blue MAX: $155.00

QUALITAS®

7101 Wisconsin Avenue ◆ Suite 1386 ◆ Bethesda, MD 20814 ◆ Telephone 301-907-6700

Circle 41 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Objective dBASE programmers try C++

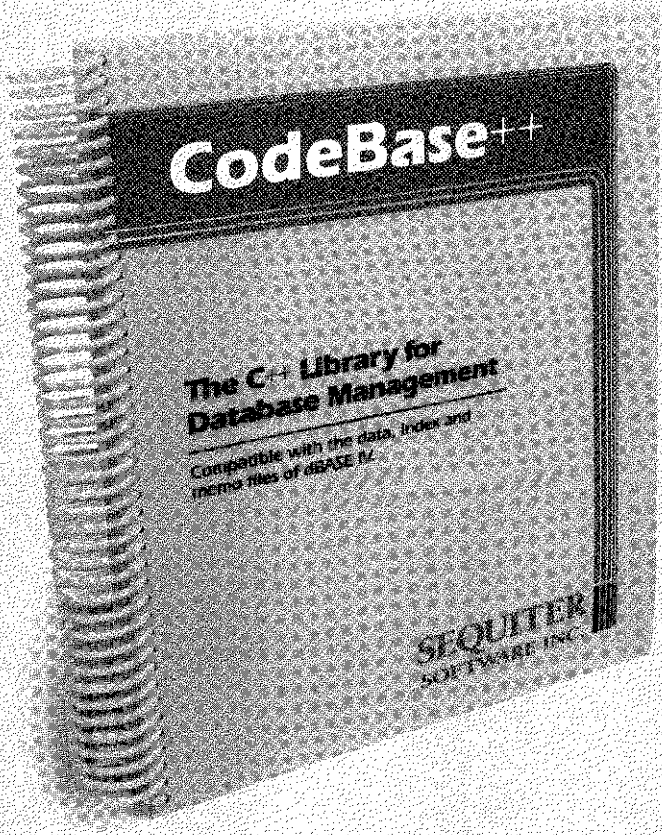
Use CodeBase++ with C++ to build multi-user, dBASE compatible programs. Take advantage of object orientated programming to quickly create maintainable, readable software.

dBASE Compatible

Use CodeBase++ classes to access and change the data, index and memo files of dBASE IV and III. Use other classes to manipulate strings, sort at high speed, perform date arithmetic, evaluate dBASE expressions, manipulate linked lists and manage memory.

Portable

Port your programs between DOS, Microsoft Windows, and OS/2. Keep all the profits as you distribute your programs royalty free.


CodeBase++
The C++ Library for Database Management
Compatible with the data, index and memo files of dBASE IV.
SEQUITER SOFTWARE INC.

Fast

Watch as your C++ programs execute much faster than corresponding dBASE, Clipper or FoxPro programs. Be surprised at the small executables and low memory requirements. Examine the C++ source code, which is completely included, to learn good object orientated programming techniques.

Order Today

Order today at $295 and discover why Sequiter Software Inc. and most dealers offer a 60-day money back guarantee.

SEQUITER SOFTWARE INC.

Call (403) 448-0313

Fax (403) 448-0315

P.O. Box 5659, Station L, Edmonton, Alberta, Canada T6C 4G1

Circle 42 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

the MRU chain. Next, it reads the desired VM block from the disk into the VM RAM block occupied by the VM block it has just swapped out to the disk, updates the *location* and *dirty* fields in its **vmheader**, and places the header at the front of the MRU chain. Then it takes the VM RAM block address, adds the offset (from the original token), and returns the address from the caller.

THE MEMORY ALLOCATOR

The allocator need only manage *free* memory, not memory that has already been allocated. Notice that the VM block structure in Listing 1 contains the member *freehead*, which is an *int*. (Remember the tokens? The position element of the token is 16 bits.) This is the offset in the VM block of the next free chunk of memory. Each *free* block contains a structure with two pieces of information: the location of the next free block and its size (see Listing 1).

Let's say the allocator gets a request for 500 bytes. It will check each VM block for 502 bytes. (The length of the allocated block is stored in the two bytes immediately preceding the block.) The allocator searches for free memory by going down the free chain for each VM block. The head of the free chain in the **vmheader** structure is **free**. It contains an offset in the VM block of a **freeblk** structure. The **vmheader** structure has a field named *maxfree*. In the interest of speed, the allocator maintains this field with the largest free memory chunk available in the VM block. That way the allocator doesn't have to search through VM blocks unnecessarily.

If the allocator finds a block of sufficient size, the most efficient way to allocate the memory is to truncate the free memory chunk. The allocator then takes the offset of the free block, adds the size of that free block, and subtracts 502. It subtracts 502 from the *size* field in the free block and updates the maximum free chunk size in the header. By combining the number of the VM block where the memory chunk was found with the offset of the chunk within the VM block, the memory allocator creates a token. This new VM token is then returned to the caller.

FREE CHUNKS AND CHAINS

Freeing memory is a little more involved. The primary goal is to add the freed memory back into the free chain for a block; freeing memory also ensures that the memory blocks do not fragment into ever-smaller pieces.

When a token is freed, it is again split into the header number and the offset components. The allocator determines whether the freed memory chunk is adjacent to another chunk of free memory; if so, it combines them to ensure that the block does not become fragmented. Because the free chain for a VM block is sorted in ascending order, it is necessary to insert the newly freed block into the free chain in the proper position. By doing so, the allocator can calculate, for each free chunk in the free chain, whether the chunk is at the end of the current free chunk. If so, it simply adds the size of the new chunk to the current chunk.

If the freed chunk is not adjacent to another memory block, then a **freeblk** structure is placed in the **free** chunk, and the **free** chunk is linked into the block's **free** chain.

However, doing this search-and-combine-or-insert process for every free request can be time-consuming. Therefore, the **vmheader** structure contains another member: **qfree**, a pointer to a **queue** of free chunks to be properly deallocated. The allocator queues up the free requests by piling them into the queue. While the application is idle (for example, waiting for keyboard input), **qfree** can call up another VM manager

Listing 1

```
/* VM block header structure */
struct vmheader {
     int  location;      /* Current block location: 0=disk,1=RAM */
     char *addr;         /* RAM location, if in RAM              */

     int  prev;          /* MRU chain previous                   */

     int  next;          /* MRU chain next                       */
     int  number;        /* VM Block # contained in this struct  */
     int  freehead;      /* Free chain next ptr                  */
     int  qfree;         /* Queued free chain head               */
     int  allocs;        /* # allocations this block             */
     int  dirty;         /* Has block been modified?             */
     int  maxfree;       /* Maximum free block size              */
     int  memtype;       /* Block type: DOS or EMS               */
     int  windnum;       /* If an EMS block, the hardware window */
     int  pagenum;       /* If an EMS block, the software page   */
};
/* VM free block structure */
struct freeblk {
     int  next;          /* Offset of the next free chunk of mem */
     int  size;          /* The size of this free chunk          */
 };
```

Listing 2

```
/* Source code illustrating use of VM calls */

typedef unsigned long  TOKEN
typedef TOKEN          TOKLINE;
typedef TOKLINE        STRING;
typedef struct strline STRLINE;

struct strline {
     TOKLINE   next;              /* token of next line in chain */
     TOKLINE   prev;              /* token of prev line in chain */
     STRING    text;              /* token of text for this line */
 };

#define lineref(linetok)        ((STRLINE*)vmderef(linetok))
#define stringref(stringtok)    ((char *)vmderef(stringtok))
#define TOKLINE0                ((TOKLINE)0)
#define STRING0                 ((STRING)0)

/* This function takes a head line token and a tail line token
 * and scans the lines between head and tail (inclusive) for a
 * line that begins with a number sign "#".
 * Returns the token of the line if found, TOKLINE0 if not found. */

TOKLINE   linescan(headtok, tailtok)
{
    TOKLINE   line;

    for (line=headtok; line != TOKLINE0; line=lineref(line)->next) {
        if (*stringref(lineref(line)->text) == '#')
            return(line);
    }
    return(TOKLINE0);
}
```

PJ CODE

All code in this issue is available through our bulletin board services or directly from *PJ* on disk. For information see pages 72 and 73.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

function, **vmemfree**, to perform this defragmentation function in the background on one freed item at a time.

When the allocator runs out of memory in the blocks, it may either allocate a new VM block (as described above) or just check to see whether there is freed memory in the queues. In either case, if no more memory is available, the allocator returns (TOKEN)0.

The terminator closes and deletes the virtual-memory swap file (and the VM block header swap file as well, if it was implemented) and returns.

Listing 2 shows some sample source code that uses the structures and methods outlined for the VM manager. The listing also shows some macros that facilitate the use of this model.

POINTS TO PONDER

Selecting the virtual-memory system best suited to your application is an important task. You must balance speed, development time, maintainability, and portability when retrofitting an application with an ASVM. In any event, it is often difficult to decide on the right ASVM. Some virtual-memory managers are available in source code to help you get a head start (these are generalized ASVMs, of course), but something as fundamental as a VM manager had better work properly; if it doesn't, or if you want to customize it, you will probably find it easier to work on your own code instead of someone else's.

What, then, is the most realistic way to develop an application using an ASVM? In retrofitting an application with virtual memory, it is often best to use a generalized VM manager like the one detailed above. The more specifically you tailor your VM manager to perform a particular task (in this case, text editing), the more it must integrate with and pervade your application. This condition makes the VM manager harder to implement, maintain, and port.

If you are starting from scratch and if your application's virtual-memory demands are not too excessive, a specialized ASVM may be the way to go. It can be straightforward to implement and give you good performance.

Remember, you are going to pay a performance penalty for enlarging your application's available memory in any virtual-memory system. In software development, the trade-off between speed and space is virtually axiomatic.

Parallel Programming for "C"

INTERWORK™

A Concurrent Programming Toolkit

Interwork is a "C" program library which allows you to write your programs as a set of cooperating concurrent tasks. Very useful for real-time applications, simulation, and parallel programming.

New version 3.0 now available

FEATURES

- Unlimited number of tasks
- Fast context switching

NEW! • Optional time sliced scheduling

- Full inter-task communication

NEW! • Enhanced interrupt and trap handling

NEW! • Expanded user's guide with examples

Interwork is available for the following systems:

IBM PC, XT, AT	PC-DOS 2.0 or later	$189
IBM PC AT	XENIX 286	$239
Intel 386	UNIX V.3/386	$329
DEC VAX™ SUN 3,4	UNIX 4.2BSD	$329

For more information please contact:

Block Island Technologies

Innovative Computer Software

15455 NW Greenbrier Parkway, Suite 210
Beaverton, Oregon 97006
(503) 690-7181 FAX (503) 645-7732

Circle 43 on Reader Service Card

PERISCOPE

Professional Software and Hardware-Assisted Debuggers for 80386 & 80486 Systems

Periscope/EM and 386MAX team up to keep Periscope out of the lower 640k...so you can have both quality memory management and a resident debugger.

If you need a debugger but have no room left in the lower 640K of your 386™ or 486™ system, Periscope now has a new SOFTWARE-ONLY solution for you—Periscope/EM.

- Periscope/EM takes advantage of the extended memory you already have, rather than using any memory in the lower 640K.
- No runaway program can corrupt Periscope/EM because it's write-protected.
- You don't have to use a slot since there's no board to install.
- You can add an optional break-out switch to recover from crashes.
- Periscope/EM costs only $295, half the cost of Periscope Model I, with most of the same functionality.

Periscope/EM requires 386MAX® or BLUEMAX™ version 5.11 or later, a 386 or 486 system with about 300K of extended memory, 32K of memory between 640K and one megabyte, DOS 3.0 or later.

New Debugging Tools

"I needed a means to debug interrupt handlers where I could really see what was going on. The hardware trace buffer is great. I was able to debug code in 3 days that I have been trying to debug for months!"

...writes Peg Sestrich with Prime Computer, on why she chose Periscope Model IV.

If you need more than your current software-based debugger gives you, Periscope Model IV, with these advanced NEW features, may be just what you're looking for.

- Supports 386 and 486 systems running up to 33MHz with its ICE-like capabilities.
- Real-time hardware trace buffer holds up to 16K CPU events.
- Enhanced analysis and display of trace buffer makes you more productive.
- Periscope/EM functionality, built in at no extra cost, saves you money.

Just call toll-free 800/722-7006 for details or to order.

The Periscope Company, Inc.

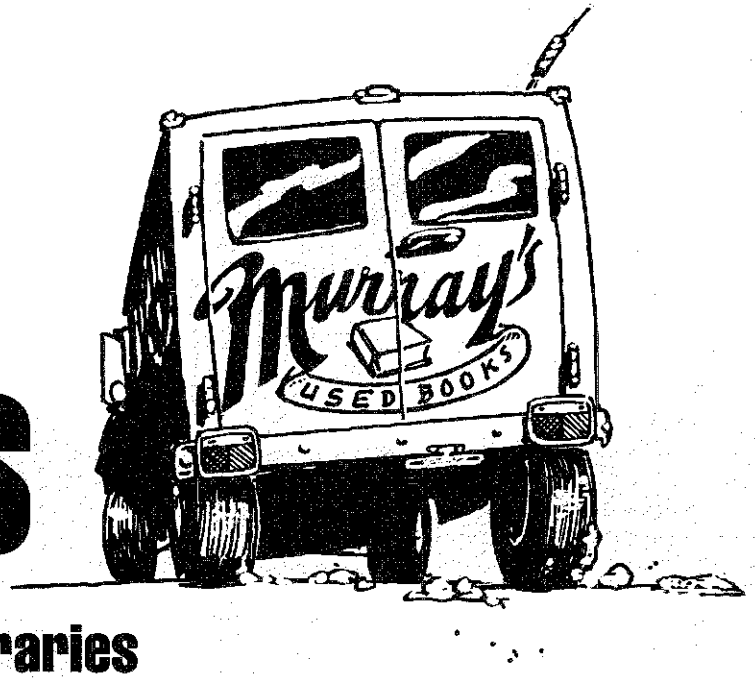
1197 PEACHTREE ST.
PLAZA LEVEL
ATLANTA, GA 30361
404-875-8080
FAX 404-872-1973

Real-time Periscope MoveI IV, shown with new 33MHz board and 486 pod.

Circle 44 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

BASIC libraries



Managing personal link libraries with reusable code

By Murray L. Lesser

In spite of what you may read in the programming press, "reusable code" is not a recent invention. The "reusable subroutines" concept was described by John Mauchly in 1947 [Reference 1]; four years later, its practical application was featured in the first textbook on computer programming.[2]

In the 1950s, when I was still earning my living by programming, we kept our personal library modules in source code that was stored as decks of punched cards. They were merged into the application program's source deck before it was link-loaded into the system. Those modules of reusable code were a natural by-product of good "bottom-up" program design.

Just about the time that it became easy to store a personal library as a single file of linkable object modules, the reusable-code concept apparently vanished. From the start of structured programming until the recent popularization of object-oriented programming (OOP) languages, new programmers were taught only the "top-down" design process, which produces code that's not reusable. It's possible that the only professional programmers writing reusable code during the interregnum were those either writing the "standard" compiler libraries or churning out add-on libraries.

The hOOPla may have a useful side effect: unspoiled programmers might learn that it is "bottom-up" programming—not the language used—that leads to reusable code. Then they'll want a better way than source-code modules to maintain a working personal library. They might even reinvent the personal link library.

A personal library needs an equivalent link library for each compiler model the programmer uses, along with a "declaration" (header) file for each source-code module using any of the library procedures. BASIC 7 (BC 7) programs running in real mode (under DOS) require as many as four LIB files (for near and far strings, each with emulator and alternate floating-point arithmetic) in addition to the $INCLUDE header file.

In what follows, I describe two utilities: DOLIB.BAS and PBBLBAS. These can be used to automatically add or replace modules in a set of BASIC 7 personal link libraries, and to edit the header file to fit. Because DOLIB is based on my personal-library enhancement of the RUN statement, I also describe the library modules that are used for that purpose, FINDFILE.BAS and COMMAND.ASM.

Murray L. Lesser is a retired IBM engineer. He has written three books about enhancing various Microsoft BASIC compilers with assembled modules. The most recent, Advanced QuickBASIC 4.0, *was published by Bantam.*

ENHANCING RUN

As furnished by Microsoft, the RUN statement in compiled BASIC can load and transfer control to any EXE or COM program that is fully specified (including drive and directory) as an argument to the statement. (COM files must also have an explicit extension; otherwise, an EXE file is assumed.) But if the requested file is not in the specified directory, RUN cannot find it.

The fix for this "file not found" limitation is the pair of compiled functions contained in the module FINDFILE.BAS (Listing 1). **findfile***(filespec)* looks for its argument file first in the specified directory (it may be the default), then in the directories given on the environment PATH string. If **findfile** finds the file, it returns the complete *filespec*, including the path to its directory; otherwise, it returns a null string. The other function, **makname***(filespec)*, deletes the "path" information from a complete file specification, returning only the filename and extension.

RUN, as furnished with compiled BASIC, has another drawback: the only way to let the child program inherit a command-line argument is directly from the parent program's command line. RUN uses a "load overlay" version of DOS's EXEC function, loading the child program's executable code and data segments into memory immediately above the parent's program segment prefix (PSP). Two changes are made in that

ILLUSTRATION BY JIM CARPENTER

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PSP before control is passed to the child program: the parent's environment space is copied into highest available memory, adding a little "growing space," and the address pointer at PSP offset 2Ch is changed accordingly. The "top-of-memory" pointer at PSP offset 2 is also changed to block off the memory used by this environment space.

If the parent program was compiled to use the runtime module (RTM), which means it was compiled without the "/O" switch, this top-of-memory address is reduced to protect the RTM as well. A little loading time would thus be saved if the child program used the same RTM, but available memory for any other child program would be reduced by about 70K.

This maneuvering gives the child program any "rest-of-the-command-line" arguments that were loaded into the top half of the parent's PSP before control was passed to the child. Thus, if you want to feed command-line information to the child, you must modify the parent's PSP before calling the RUN statement. The subroutine in COMMAND.ASM (Listing 2) performs this modification.

COMMAND needs to find both the string and the PSP, because **COMMAND**(*string*) then copies the value of the string from BASIC's string space into the running program's PSP in the proper format. Two BASIC 7 mixed-language compiler library functions—**StringLength** and **StringAddress**—are used to find the string representation in whatever string space it occupies. The undocumented DOS function **51h** finds the segment address of the parent program's PSP. If your programs will never be run under a DOS version earlier than 3.0, you can substitute the equivalent documented function **62h**.

LINK LIBRARIES AND UPDATES

In its full glory, BC 7 offers too many choices. You have the traditional compiled BASIC choice between a stand-alone executable file and one requiring the runtime module. You can configure your system at SETUP time to allow compile-time choices of either of two methods of performing floating-point arithmetic. You can choose whether you want the traditional near-string space or multiple far-string spaces, and whether your final program will run under real (DOS) or protected (OS/2) processor mode.

The choice of which link library to use is made by command-line switches when the compiler is called. If the appropriate link library is not available at link time, LINK will fail with a warning message. All compiled modules to be linked, including those in any personal (or add-on) library, must have been compiled using the same arithmetic-, string-, and mode-determining switches. (If they were not, you will get the runtime error message "Error during runtime initialization," and the program will die before it gets started.) However, you need not specify whether an RTM will be used when compiling a personal library module. Because my BASIC program-development files are configured only for emulator arithmetic and for real-mode processing, I maintain only two personal libraries—one for near-string compilation and one for far strings.

DOLIB.BAS (Listing 3) is an automatic library update utility. When given a module name, it updates my two personal libraries and my declaration header file by replacing (or adding) the module and by editing the DECLARE statements in the header file—all from the module source-code file. Two library index (text) files are produced as by-products.

Writing my own utilities allows me to establish the naming conventions. For DOLIB, the near- and far-string libraries are PB.LIB and PBF.LIB, respectively. The corresponding index files have the extension IDX, and the header file is PB.BI.

In order to edit PB.BI automatically, module source code contains its DECLARE statements as remarks that are coded between square brackets. No space is allowed between the left square bracket and the word "declare." The closing right square bracket must be on the same line as the left bracket, and only one such statement can appear on that line. Procedure declarations are case insensitive.

The original library files are stored in the directory pointed to by the LIB environment string. The original header file resides in the directory pointed to by the INCLUDE string. DOLIB updates copies of those files that are moved to the default directory and leaves the originals unscathed.

DOLIB operates by first writing, and then RUNning, a self-erasing batch file that does most of the work. (A sample DOLIB $$$.BAT file is shown in Figure 1.) If you run your development system with a virtual disk as the default drive, writing and executing a batch file is much faster than executing the required processes via multiple SHELL statements from within the program.

DECISIONS, DECISIONS

While writing the batch file, DOLIB makes a number of decisions. If any files to be updated are not already in the default directory, they are copied. If the module is to be compiled, it is "read" first to see whether local error trapping is used; if so, it is compiled with the "/E" switch. BASIC modules are compiled twice—once for each library. Assembled modules are assembled only once.

The PRINT #1 statement that writes the self-deleting last record in the output $$$.BAT file ends with a semicolon, so there is no line-end symbol at the end of that file to produce a "batch" error message. Instead of using FINDFILE, DOLIB finds the command processor (the argument to the RUN statement) from the COMSPEC environment string.

One batch file task is to run the header-file editing utility PBBI.BAS (Listing 4). PBBI again reads the module source file, looking for the coded DECLARE statements. It builds two string arrays; one contains the declaration statements and the other contains the corresponding procedure names. Any type-defining suffix attached to a function name is deleted, allowing former subroutine procedures to be replaced by functions having the same name.

FIGURE 1: Sample DOLIB $$$.BAT file

```
@ECHO OFF
COPY C:\LIB\PB.LIB
COPY C:\LIB\PBF.LIB
COPY C:\HEADER\PB.BI
BC C:\SOURCE\FINDFILE.BAS;
IF ERRORLEVEL 1 GOTO END
LIB PB-+findfile,PB.IDX;
IF ERRORLEVEL 1 GOTO END
BC C:\SOURCE\FINDFILE.BAS/Fs;
IF ERRORLEVEL 1 GOTO END
LIB PBF-+findfile,PBF.IDX;
IF ERRORLEVEL 1 GOTO END
DEL findfile.OBJ
PBBI C:\SOURCE\FINDFILE.BAS
:END
DEL $$$.BAT
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LISTING 1

```
                        FINDFILE.BAS
'    Two functions to be linked to programs compiled with BC version 7.
'    MAKNAME(<filespec>) [DECLARE FUNCTION MAKNAME$ (A$)] returns the
' <filename.ext> portion of <filespec>; <filespec> can be either a string
' variable or string literal.

'    FINDFILE(<filespec>) [DECLARE FUNCTION FINDFILE$ (A$)] locates a file
' stored in the specified directory or in a directory on the PATH environ-
' ment string. FINDFILE returns a string containing the full <filespec>.
' If the input argument contains "wildcards," FINDFILE returns the
' expanded <filespec> of the first file that meets the input specifi-
' cation.  If the requested file cannot be found, FINDFILE returns a
' null string.
'
'              Rewritten by M. L. Lesser, April 8, 1990
' Based on BC version 4.0 subroutines written for "Advanced QuickBASIC
' 4.0," published by Bantam Books, 1988.

'               Compiled with Microsoft BC version 7.1

    DEFINT I-K
    DEFSTR F,M,P,T

FUNCTION MAKNAME(FILENAME) STATIC
    LET J = INSTR(FILENAME,"\")          'Separator in <filespec> "path"
    LET K = 0                            'Place mark
    WHILE J <> 0
        LET K = J                            'K will be location of
        LET J = INSTR(K+1,FILENAME,"\")      ' last "\" in FILENAME
    WEND
    IF K = 0 THEN LET K = INSTR(FILENAME,":")  'If no path, try drive
    IF K <> 0 THEN
        LET MAKNAME = MID$(FILENAME,K+1)
    ELSE
        LET MAKNAME = FILENAME
    END IF
END FUNCTION

FUNCTION FINDFILE(FILE) STATIC
    LET PATH = ENVIRON$("PATH") 'Get PATH environment string
' Capture any FPATH from input FILE in case hit on first directory:
    LET FPATH = UCASE$(LEFT$(FILE,INSTR(FILE,MAKNAME(FILE))-1))
    LET TEMP = FILE                      'Local variable
' Now find the file:
    WHILE LEN(DIR$(TEMP)) = 0            'As long as file not found
        LET I = INSTR(PATH,";")              'Separator in PATH string
        IF PATH = "" THEN                    'If no more directories
           LET FINDFILE = ""                     'Error signal and forced
           EXIT FUNCTION                         '  return from function
        END IF
        IF I <> 0 THEN                       'If more than one "path"
           LET FPATH = LEFT$(PATH,I-1)           'Try next one and
           LET PATH = MID$(PATH,I+1)             ' strip it off string
        ELSE                                 'This is last one on string
           LET FPATH = PATH                      'So use it
           LET PATH = ""                         '  and set PATH at null
        END IF
        IF RIGHT$(FPATH,1) <> "\" THEN       'If not a root directory
           LET FPATH = FPATH + "\"               '  add trailing "\"
        END IF
        LET TEMP = MAKNAME(TEMP)             'Delete old "path,"
        LET TEMP = FPATH + TEMP              '  insert new value,
    WEND                                     '  and try again
    LET FINDFILE = FPATH + DIR$(TEMP)
END FUNCTION
```

LISTING 2

```
        TITLE   Command Line
        .MODEL  MEDIUM
;                       COMMAND.ASM
;   COMMAND(<string>) [DECLARE SUB COMMAND (A$)] is an assembled
; subroutine to be linked to programs compiled with Microsoft BC v.
; 7.0 (or later). It inserts "the rest of the command line" in the PSP
; before starting another program with the "RUN <filename>" command and
; operates for programs compiled in either the "near string" or "far
; string" mode.
;   Call with:  CALL COMMAND(<string>)
;       where <string> is the argument string to be returned by the
;       new program.
;   If LEN(<string>) > 125, a null argument string will be inserted
; instead.
```

continued

BASIC 7 STUB FILES

In addition to redesigned link libraries for greater granularity, the BASIC 7 package includes a number of "stub files" that can be linked to object programs. These files can reduce the number of "just in case" BASIC support functions that are usually linked in automatically.

The useful stub files are those that link to stand-alone programs (those compiled with the "/O" switch). These files can reduce the size of the final program by replacing unused support code that would otherwise be linked into the program. Discussed here are four BASIC 7 stub files: NOCOM.OBJ, NOLPT.OBJ, NOFLTIN.OBJ, and NOTRNEMR.LIB. The first three are particularly useful with text processing programs such as DOLIB and PBBI.

Whenever a file is OPENed with a variable for a filename, the compiled program calls for the communication and printer support to be brought in from the link library, just in case that file turns out to be a device. These called for items include the start-up code to generate buffer space for each installed serial and parallel adapter. (This unfortunate state of affairs is a direct consequence of the misbegotten notion, imported into DOS 2.0, that operating systems should support character devices in the same manner as they do files.) For many programs, you can save about 3,000 code bytes by linking to NOCOM.OBJ as one of your linker "object" files. If you aren't using the printer, NOLPT.COM could save another 1,200 bytes.

The BASIC 7 compilers request floating-point support whenever the source code contains a **VAL**, **INPUT**, or **READ** statement—again, just in case. Unfortunately, floating-point support is also requested by a **LINE INPUT** statement, even though the actual input is always a string. If you have no **VAL** statements and no **INPUT** or **READ** statements that involve floating-point values, and if your program doesn't otherwise require floating-point support, you might save about 13,000 bytes (emulator mode) by linking to NOFLTIN.OBJ. If you do require floating-point support for other source-code statements, you might still save about 1,400 bytes. If you are using emulator mode and need the floating-point support but do not have any transcendental functions in your code, you can also link to NOTRNEMR.LIB, removing another 1,700 bytes. Note that NOTRNEMR.LIB is furnished as a library of several modules; you must use the full name, NOTRNEMR.LIB, in the "object file" portion of the linker command.

BASIC 7's treatment of the **VAL** statement is anomalous. It always drags in floating-point support even when it's linked to NOFLTIN.OBJ. Microsoft changed **VAL** between BC 7.0 and BC 7.1, aggravating the problem: if your program uses the **VAL** statement but does not also include some form of **INPUT** statement, **VAL** becomes inoperative (returning only a zero) if the program is compiled with BC 7.1 using the "/O" switch and is linked to the BC 7.1-furnished version of NOFLTIN.OBJ.

If used correctly, stub files can be very useful. Improper use can make your program longer or even make it inoperative. Try before you buy.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

After building the two arrays, PBBI looks in the old PB.BI file for any procedure name that was declared in the new module. If it finds one, it deletes the corresponding old declaration and appends the new module declarations to the end of the file.

When the batch file finishes, the original and the updated files are available in the default directory. The old library files now have a BAK extension, and the old header file is now named PB.BKE.

If the files to be updated don't already exist, a slight "chicken and egg" problem occurs when running the update utilities for the first time. DOLIB will create the two new libraries in the default directory if there are no previous versions, but if PBBI can't find an old PB.BI file to update, it will die with a "file not found" error. The easiest fix for this is to write a dummy PB.BI file from the keyboard, with the dummy file containing only a line-end symbol. Once the library and header files contain the FINDFILE and COMMAND modules, you can edit DOLIB.BAS to delete the three procedure declarations and substitute the line

```
' $INCLUDE: 'PB.BI'
```

The recompiled DOLIB can now be linked to PB.LIB rather than to the individual procedure object modules. If the personal library is named in the "library" field of the linker command, LINK extracts only those modules that resolve externals.

ALWAYS KEEP AN EYE OUT

Once the utilities are in final form, using them is relatively bulletproof, although you should watch the display for the following anomaly in Microsoft's LIB utility. If the "replacement" module isn't already in the library, LIB issues a "module not in library" warning message before the LIB sign-on message. This is normal when you add new modules to the library, and the message can be ignored. However, if LIB issues the warning "symbol defined in *modulename*, redefinition is ignored" after the sign-on, you are in trouble; the module you have added contains a procedure name that duplicates one already used in another module. (Unfortunately, Microsoft didn't make this a fatal LIB error that issues an exit code.) In this case, discard the new files and correct your module before trying again.

Another situation to watch out for arises whenever PBBI cannot find the trailing square bracket for a declaration statement. PBBI will then force a BASIC "Illegal function call" error, which halts its operation. It will also produce an erroneous PB.BI file if a replacement module does not declare all of the public names used in the original. The only check for this occurrence is to peruse the updated listings of the three text files: PB.IDX, PBF.IDX, and PB.BI. I keep the most recent copies of these in my BASIC programmer's notebook.

REFERENCES

1. Mauchly, J. W. (1948). *Preparation of Problems for Edvac-Type Machines.* Proceedings of a Symposium on Large-Scale Digital Calculating Machinery, 7–10 January 1947. Cambridge, MA: Harvard University Press, pp. 203–207.
2. Wilkes, M. V., Wheeler, D. J., and Gill, S. (1951). *The Preparation of Programs for an Electronic Digital Computer.* Cambridge, MA: Addison-Wesley.


```
;               Written by M. L. Lesser, March 30, 1990
;    Based on BC version 4.0, written for "Advanced QuickBASIC 4.0,"
; published by Bantam Books, 1988.
;                 Assembled with Microsoft MASM v. 5.1

EXTRN StringAddress:FAR                 ;"Mixed language" functions
EXTRN StringLength:FAR                  ; added to BC 7.0 libraries

.CODE
        PUBLIC COMMAND
COMMAND PROC
; Save critical registers used:
        PUSH    BP
        MOV     BP,SP
        PUSH    DS
        PUSH    SI
        PUSH    DI
; Get string-argument length first:
        PUSH    6[BP]                   ;Argument for StringLength
        CALL    StringLength            ; (offset of string descriptor)
        PUSH    AX                      ;Function return in AX
; Get address of string:
        PUSH    6[BP]
        CALL    StringAddress           ;Function returns far address
        MOV     DS,DX                   ; pointer in DX:AX
        MOV     SI,AX
; Get PSP segment address:
        MOV     AH,51H                  ;Undocumented DOS call
        INT     21H                     ;  returns PSP segment address
        MOV     ES,BX                   ;  in BX
        MOV     DI,80H                  ;Start of argument pointer
; Insert string in command line in proper form:
        POP     CX                      ;String length
        OR      CX,CX                   ;Is it null string?
        JZ      NULL
        CMP     CX,7DH                  ;Is it within length limit?
        JA      NULL                    ;If not, replace with null
        MOV     AL,CL                   ;Number of bytes in string
        INC     AL                      ;Add one for leading <SP>
        STOSB                           ;Store command-line byte count
        MOV     AL,' '                  ;Insert leading <SP>
        STOSB
        REP     MOVSB                   ;Move text of <string>
DONE:   MOV     AL,0DH                  ;Add trailing <CR>
        STOSB
        POP     DI
        POP     SI
        POP     DS
        POP     BP
        RET     2
NULL:   XOR     AL,AL                   ;Forces null command line
        STOSB
        JMP     DONE
COMMAND ENDP

        END
```

LISTING 3

```
'    DOLIB.BAS updates the private libraries used with Microsoft BC
' version 7.  It adds (or replaces) named modules to the private
' searchable libraries PB.LIB and PBF.LIB by constructing and running
' a batch file.  The batch file also runs the compiled BASIC program
' PBBI.EXE to update the declaration header file, PB.BI.
'    The command-line call is "DOLIB [<path>]<modname>".
'                  <path> is required if the source-code file for
'                         <modname> is not in a directory in the PATH
'                         environment string.
'    If the source file for <modname> is <modname>.ASM, DOLIB. will be
' assembled before adding it to both libraries.  If it is <modname>.BAS,
' the file will be compiled twice: in default mode for PB.LIB, and for
' "far strings" for PBF.LIB.  If the BASIC source file contains local
' error trapping, the compilation will be made with the "/E" switch.
'    DOLIB updates in the default directory. If any of the files to be
' updated is not in the default directory, DOLIB will copy it from the
```

continued

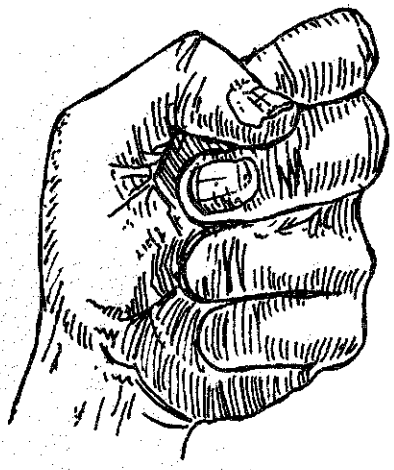
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
' directory named in the LIB or INCLUDE environment string. If <modname>
' already exists, it will be replaced by the new version, as will
' an existing SUB or FUNCTION declaration statement in the PB.BI file.
'               Written by M. L. Lesser, April 14, 1990
'                 Compiled with BC v. 7.1, switch "/O"
'        Linked to COMMAND, FINDFILE, NOCOM, NOLPT, and NOFLTIN
'                        with switches "/E/NOE"

    DEFINT I
    DEFSTR E,F,M,P,T
    DECLARE FUNCTION FINDFILE$ (A$)
    DECLARE FUNCTION MAKNAME$ (A$)
    DECLARE SUB COMMAND (A$)

' Local subroutine to write $$$.BAT command that copies file from
'   directory designated in Environment string to default directory:
SUB WRITIT(ENVIRONMENT, FILENAME)
LET PATH = ENVIRON$(UCASE$(ENVIRONMENT))
    IF RIGHT$(PATH,1) <> "\" THEN LET PATH = PATH + "\"
    PRINT #1, "COPY " PATH FILENAME
END SUB

    COLOR 2,0                           'Dealer's choice
    CLS
    PRINT TAB(20) "Private Library Update Program for BC 7"
    PRINT STRING$(80,"-")
    LET TEXT = COMMAND$                 'Get <modname> from command line
' Make sure there is a <modname> to add:
    WHILE LEN(TEXT) = 0
        LINE INPUT "Enter [<path>]<modname> for update:  "; TEXT
    WEND
    LET I = INSTR(TEXT,".")             'Remove any extension
    IF I <> 0 THEN LET TEXT = LEFT$(TEXT,I-1)
    LET MODNAME = LCASE$(MAKNAME(TEXT))
    PRINT "Adding Module " CHR$(34) MODNAME CHR$(34);
    PRINT " to private libraries PB.LIB and PBF.LIB."
    PRINT
    OPEN "$$$.BAT" FOR OUTPUT AS #1
    PRINT #1, "@ECHO OFF"               '"@" requires DOS 3.3 (or later)
' Check for private libraries and header file in default directory:
    IF LEN(DIR$("PB.LIB")) = 0 THEN CALL WRITIT("LIB","PB.LIB")
    IF LEN(DIR$("PBF.LIB")) = 0 THEN CALL WRITIT("LIB","PBF.LIB")
    IF LEN(DIR$("PB.BI")) = 0 THEN CALL WRITIT("INCLUDE","PB.BI")
' Try assembled module first:
    LET FILE = FINDFILE(UCASE$(TEXT) + ".ASM")
    IF LEN(FILE) <> 0 THEN              'Assemble it
        PRINT #1, "MASM " FILE ";"
    ELSE
        LET FILE = FINDFILE(UCASE$(TEXT) + ".BAS")
        IF LEN(FILE) <> 0 THEN
            OPEN FILE FOR INPUT AS #2       'Read first, to check for
            WHILE NOT EOF(2)                ' local error trapping
                LINE INPUT #2, TEXT         'Variable name is reused
                IF INSTR(UCASE$(TEXT),"LOCAL ERROR GOTO") THEN
                    LET FILE = FILE + "/E"
                    GOTO DONE
                END IF
            WEND
DONE:       CLOSE #2
            PRINT #1, "BC " FILE ";"
        ELSE                            'Abnormal end if not source file
            PRINT TAB(5) "Source code for " CHR$(34) MODNAME CHR$(34);
            PRINT " module not found.  Terminating program."
            CLOSE #1
            KILL "$$$.BAT"
            END
        END IF
    END IF
    PRINT #1, "IF ERRORLEVEL 1 GOTO END"        'Assemble/compile error
' Replace/Add module in PB.LIB (near strings):
    PRINT #1, "LIB PB-+" MODNAME ",PB.IDX;"
    PRINT #1, "IF ERRORLEVEL 1 GOTO END"
' If module was compiled, recompile for far strings for PBF.LIB
    IF MID$(FILE,INSTR(FILE,"."),4) = ".BAS" THEN
        PRINT #1, "BC " FILE "/Fs;"
        PRINT #1, "IF ERRORLEVEL 1 GOTO END
    END IF
' Replace/Add module in PBF.LIB (far strings):
    PRINT #1, "LIB PBF-+" MODNAME ",PBF.IDX;"
    PRINT #1, "IF ERRORLEVEL 1 GOTO END
' Clean up default directory by removing object file:
    PRINT #1, "DEL " MODNAME ".OBJ"
' Update PB.BI header file after deleting "/E" switch from file:
    LET I = INSTR(FILE,"/")
    IF I THEN LET FILE = LEFT$(FILE,I-1)
    PRINT #1, "PBBI " FILE
' Closeout
    PRINT #1, ":END"
    PRINT #1, "DEL $$$.BAT";
    CLOSE #1
    LET FILE = ENVIRON$("COMSPEC")      'Use COMMAND.COM
    CALL COMMAND("/C $$$")              '  to run $$$.BAT
    RUN FILE
```

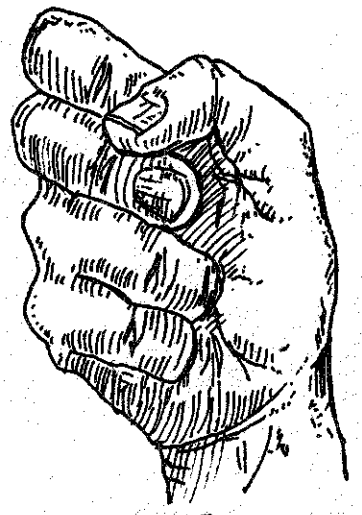
LISTING 4

```
'                                   PBBI.BAS
'   PBBI updates the declaration header file PB.BI, used with Microsoft
' BC v. 7, for a new or replacement library module. It is usually run
' by the $$$.BAT file produced by DOLIB.EXE, but it can be run as a
' standalone if PB.BI is in the default directory. When run as a stand-
' alone, call with: "PBBI <filespec>".
'    If the source file contains coded SUB or FUNCTION declaration
' statements (enclosed in square brackets), the PB.BI header file will
' be updated and the previous PB.BI renamed PB.BKE. If the source file
' does not contain coded declaration statements, the new PB.BI will
' be identical to PB.BKE.
'               Written by M. L. Lesser, April 14, 1990
'                 Compiled with BC v. 7.1, switch "/O"
'       Linked to NOCOM, NOLPT and NOFLTIN with switches "/E/NOE"

    DEFINT I-N
    DEFSTR D,F,P,T
    DIM DECLARATIONS(32), FILENAME, I, J, K, N, PROCNAME(32), T, TEXT

    OPEN COMMAND$ FOR INPUT AS #1 LEN=5120          'Source code file
    LET N = 0
' Search source file for coded embedded procedure declarations:
    WHILE NOT EOF(1)
        LINE INPUT #1, TEXT
        LET I = INSTR(UCASE$(TEXT),"[DECLARE")
        IF I THEN                                           'Found one
            LET J = INSTR(TEXT,"]")
            LET DECLARATIONS(N) = RTRIM$(MID$(TEXT,I+1,J-I-1))
            LET N = N + 1
        END IF
    WEND
    LET N = N - 1                       'Index of last used array member
    CLOSE #1
' Parse for procname (second token after "declare").  May end with
'  either a <SP> or a "(", or be the last token in the declaration.
    LET T = "%&!#@$"                    'Data type symbols
    FOR I = 0 TO N
        LET TEXT = LTRIM$(MID$(DECLARATIONS(I),8))
        LET TEXT = LTRIM$(MID$(TEXT,INSTR(TEXT," ") + 1))
        LET J = INSTR(TEXT," "): K = INSTR(TEXT,"(")
        IF J = 0 AND K = 0 THEN
            PROCNAME(I) = TEXT
        ELSEIF J = 0 AND K <> 0 THEN
            LET PROCNAME(I) = LEFT$(TEXT,K-1)
        ELSEIF J <> 0 AND K = 0 THEN
            LET PROCNAME(I) = LEFT$(TEXT,J-1)
        ELSEIF J < K THEN
            LET PROCNAME(I) = LEFT$(TEXT,J-1)
        ELSE
            LET PROCNAME(I) = LEFT$(TEXT,K-1).
        END IF
' Delete any trailing type designation symbol:
        IF INSTR(T,RIGHT$(PROCNAME(I),1)) THEN
            LET PROCNAME(I) = LEFT$(PROCNAME(I),LEN(PROCNAME(I)) - 1)
        END IF
    NEXT I
' Update PB.BI file, renaming old version to PB.BKE:
    IF DIR$("PB.BKE") <> "" THEN KILL "PB.BKE"  'Delete old back file
    NAME "PB.BI" AS "PB.BKE"
    OPEN "PB.BKE" FOR INPUT AS #1 LEN=1024
    OPEN "PB.BI" FOR OUTPUT AS #2 LEN=1024
    WHILE NOT EOF(1)
        LINE INPUT #1, TEXT
        FOR I = 0 TO N
            IF INSTR(UCASE$(TEXT),UCASE$(PROCNAME(I))) THEN GOTO FOUNDIT
        NEXT I
        PRINT #2, TEXT                  'If not in this module
FOUNDIT:
    WEND
' Add new declarations at end of PB.BI:
    FOR I = 0 TO N
        PRINT #2, DECLARATIONS(I)
    NEXT I
    CLOSE
END
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



BIOS compression



Using PS/2 BIOS compression as a tool for DOS memory management

By John E. Swanke

As MS-DOS applications and TSRs grow fat with features, the 640K boundary becomes more stifling. The straightforward solution to this situation would be to upgrade to a boundless (for now) operating system such as OS/2. But users are hesitant to take on expensive upgrades and slow learning curves. The advent of paging and address translation in the Intel 80386 has spawned the idea behind memory managers, which is to fill in the empty areas. The available areas above the 640K boundary, between hardware adapters, and below the BIOS (Basic Input/Output System) can be used to store expanded and extended memory, as well as your TSRs (see Figure 1).

These filled-in areas are called Hi-DOS memory (the High Memory Area under XMS). Hi-DOS memory is created by using paging on the 386 processor and then mapping extended memory above 1MB (or expanded memory) to the unused logical segments between A000h and E000h.

While TSR programmers have been making their applications fat, so have BIOS programmers. IBM's PS/2 BIOS without advanced BIOS support for OS/2 is 64K; with this enhancement, it's 128K. Add another 128K for a VGA and 64K for an EMS page, and there's almost no room left for TSRs. The need for more creative memory management ideas under DOS is obvious.

Fortunately, not all of a BIOS's ROM is used, so it too can be filled in with expanded and extended memory via paging and address translation. IBM's PS/2 BIOS has the following unused areas:

- advanced BIOS—not used by DOS;
- power on self-test (POST)—POST code isn't used after power-up;
- BASIC subsystem—32K of IBM BIOS can be replaced with GW-BASIC;
- filler areas—all burned-in ROMS have spare areas to grow in.

Unlike hardware addresses, however, these BIOS software areas have no standard starting or ending addresses. So how do you tell your memory manager where to fill in?

THE QUARTERDECK APPROACH

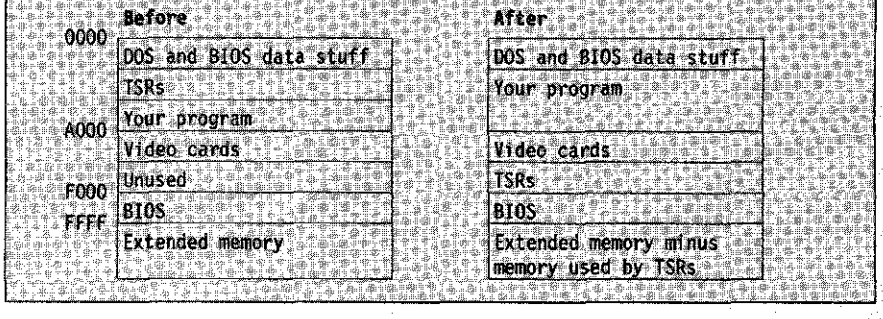
Quarterdeck's Expanded Memory Manager (QEMM) 386 has an analysis feature that finds unused areas by monitoring which ROM addresses are used. After you've run all of your desired applications and have used all of your hardware at least once, you get an analysis of your unused areas, as shown in Table 1 for my PS/2 70 system.

I got an extra 76K by appending the following line to my DEVICE= statement using QEMM:

```
/INCLUDE=E900-F1FF/INCLUDE=F300-F3FF/INCLUDE=F500-FDFF
```

Unfortunately, there are many drawbacks to this approach. First, 80386 paging

FIGURE 1: TSRs before and after a memory manager



John E. Swanke is a recent graduate from the minicomputer world. He wrote RJSwantek, Inc.'s Dis•Doc Professional Disassembler, used to create the BCF files described in this article. He can be reached at 178 Brookside Road, Newington, CT 06111, or on CompuServe at 73734,3640.

ILLUSTRATION BY JIM CARPENTER

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

works only in 4K chunks; thus, if a 3K chunk were at E400, it couldn't be used. Also, the extra 76K is not contiguous, and most TSRs can't handle noncontiguous memory. (If you have small TSRs, this may not be a problem.) Finally, the extra 76K may not be reliable; some of the "unused" areas may be used later with new applications or hardware, causing your system to act weird or even crash.

THE QUALITAS APPROACH

Qualitas's 386MAX memory manager looks for unused areas, specifically 4K ROM filler areas. These are areas of repeated 0FFh or 0CCh bytes. (IBM generally uses 0CCh, which is the breakpoint interrupt; Phoenix uses 0FFh, which is a ROM data byte before any program is burned in.)

Some of the "unused" areas may be used later, causing your system to act weird or even crash.

But Qualitas had an even better idea with its newest IBM PS/2 memory manager, BlueMAX: it accurately locates all unused BIOS areas and compresses them out, leaving one big contiguous chunk of Hi-DOS memory (Figure 2). BlueMAX works by copying all used areas of the BIOS from ROM memory into a contiguous chunk of RAM memory, leaving out whatever's unused. Then the BIOS is relinked on the fly, and the interrupt jump table starting at 0:0 is revectored into the new compressed BIOS.

Relinking means fixing up the relative addressing in JMP, CALL, and MOV operations. Revectoring means fixing up INT 6 and others to point to their new resting spots in the compressed BIOS. Which addresses should be fixed up, which areas compressed out? Each BIOS is different. Just as a word processor has a different configuration file for every printer it supports, BlueMAX has a different BIOS

TABLE 1: QEMM analysis results

Area	Size	Status
0000–E8FF	932K	OK
E900–F1FF	36K	Include
F200–F2FF	4K	OK
F300–F3FF	4K	Include
F400–F4FF	4K	OK
F500–FDFF	36K	Include
FE00–FFFF	8K	OK

AMAZING BIOS FACTS

Although assemblers should generate small, fast code for a ROM program, BIOS programmers didn't use optimizing assemblers until recently. You'll still find plenty of

```
JMP SHORT LABEL
NOP
```

or

```
MOV AX,2312h
AND AX,0FFF0h
```

or, using the 81h opcode instead of 83h in the following (and thereby losing a byte)

```
AND AX,3
```

Some BIOSes have three types of hard disk support (INT 13h), whereas others have none. Although some BIOS routines, such as those that support INT 40 (floppy driver), are almost identical from BIOS to BIOS, INT 13h's (hard-disk driver) routines vary wildly.

Although IBM's 386 PS/2 BIOS is twice the size (128K) of the AT BIOS, ostensibly to add the advanced BIOS features, the advanced BIOS requires only about 28K. Extensive features to other BIOS functions were also added—programmers tend to fill all available space.

Although there is one major POST area, each BIOS service (such as INT 10h and 13h) also has a built-in mini-POST. (BlueMAX compresses out these areas.)

Even though these 128K BIOSes are sometimes vastly different in content, they still compress down to within the same approximate size (40–44K).

TABLE 2: BIOS entry points

Interrupt	Entry Point	Description
2	fe2c3	NMI
5	fff54	Print screen
8	ffea5	Timer tick
9	fe987	Keyboard interrupt
e	fef57	Floppy-disk controller
10	ff065	CRT support
11	ff84d	Get equipment list
12	ff841	Get usable memory
13	*	Hard-disk services
14	fe739	Serial-port services
15	ff859	General services
16	fe82e	Keyboard services
17	fefd2	Printer services
18	*	BASIC
19	fe6f2	Warm boot
1a	ffe6e	Real-time clock services
1c	fff53	18-ms timer
20–3f	–	MS-DOS
40	fec59	Floppy interrupt
41,46	*	Hard-disk param tables (sometimes)
43	*	EGA video param table (sometimes)
70	fe98c	IRQ 8 – real-time clock
76	*	IRQ 14 – hard-disk controller
–**	ffff0	Cold start/POST

* These interrupts are usually set on the fly at POST time.
** Any unlisted interrupts up to interrupt 77h are set to the spurious hardware interrupt, D11. This address can be located by looking in the interrupt jump table for a common pointer, which can usually be found in the higher interrupts, such as those between 48h and 4fh.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

compression file (BCF) for every version of a PS/2 BIOS it can compress.

MAKING YOUR OWN BCF FILES

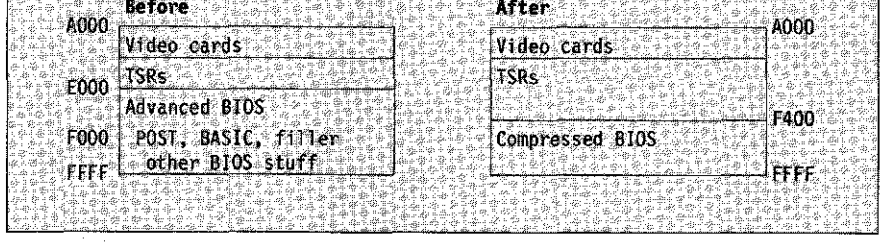
BCF files are created by using a specially modified disassembler. First, the target BIOS is disassembled to identify where each BIOS function is processed and to locate opcodes that use relative addressing. This disassembly must be 100% accurate. Next, the BIOS functions to be eliminated are identified. These include INT 15h functions 4 and 5 (the advanced BIOS) as well as INT 18h (the BASIC subsystem). Finally, the BCF file is filled with **move** and **fixup** commands for this BIOS. This BCF file is then added to BlueMAX's DEVICE= line as follows:

```
/BCF=@6549.BCF
```

BCF testing starts by acquiring the IBM PS/2 that has the target BIOS you want to support. After a BCF file is created, a test program is run on the target PS/2. The test program executes all BIOS functions and subfunctions while a record is kept of every executed opcode. This is done for both the compressed and the uncompressed BIOS using a hardware-assisted debugger (see Brett Salter's article in this issue). For example, INT 10h, function 1 opcodes are compared against the opcodes executed in the compressed version:

```
Uncompressed BIOS:        Compressed BIOS:
   cmp  al,1                 cmp  al,1
   je   1e2310               je   1f5241
   xor  ah,ah                xor  ah,ah
```

FIGURE 2: BIOS before and after compression



Notice that allowances are made for relative address changes.

To date, it has required fewer than two dozen BCF files to support all of IBM's 386 PS/2s. Although BIOS compression can also be done with non-PS/2 BIOSes, hundreds of BCF files would be required. Phoenix seems to create a new BIOS for just about every PC vendor it sells to.

DISASSEMBLING THE BIOS

Figure 3, which shows a general map of a PS/2 BIOS, displays the results of

Total control with LMI FORTH™

For Programming Professionals: an expanding family of compatible, high-performance compilers for microcomputers

For Development:
Interactive Forth-83 Interpreter/Compilers for MS-DOS, OS/2, and the 80386
- 16-bit and 32-bit implementations
- Full screen editor and assembler
- Uses standard operating system files
- 500 page manual written in plain English
- Support for graphics, floating point, native code generation

For Applications: Forth-83 Metacompiler
- Unique table-driven multi-pass Forth compiler
- Compiles compact ROMable or disk-based applications
- Excellent error handling
- Produces headerless code, compiles from intermediate states, and performs conditional compilation
- Cross-compiles to 8080, Z-80, 8086, 68000, 6502, 8051, 8096, 1802, 6303, 6809, 68HC11,34010, V25, RTX-2000
- No license fee or royalty for compiled applications

Laboratory Microsystems Incorporated
Post Office Box 10430, Marina del Rey, CA 90295
Phone Credit Card Orders to: (213) 306-7412
FAX: (213) 301-0761

Circle 48 on Reader Service Card

WHY BE LIMITED BY 640K?

XMEM
VIRTUAL MEMORY LIBRARY FOR 'C' AND ASSEMBLER PROGRAMMERS

If you've got a memory hungry application you need XMEM. XMEM completely manages conventional, expanded, and extended memory for you with a single, simple interface.

- Manages all memory, swaps to disk when memory is exhausted.
- Optionally emulates Windows, Macintosh, or OS/2 memory managers.
- Use alone or in conjunction with Malloc.
- Eliminates memory fragmentation problems.
- Supports Microsoft C, Turbo C, C++, Zortech C, C++, and others.
- Comprehensive examples, source code included!
- 30 day unconditional guarantee.

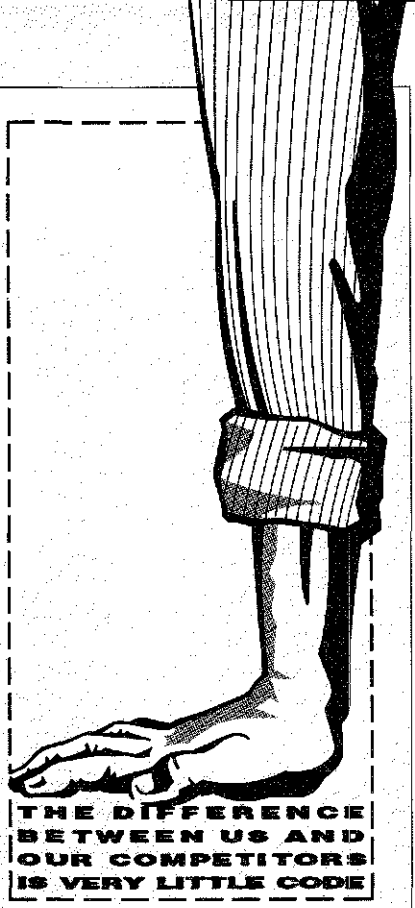
$149

TO ORDER CALL:
1•800•755•7973
MASTERCARD • VISA • COD

KANDU, INC. • 16 Cannon Dr. • Hamilton, NJ 08690 U.S.A.
Phone: 609-587-7973 Fax: 609-587-9412

Circle 45 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



Magic TSR Toolkit

Write complete memory resident programs that require as little as 2K of RAM by swapping to EMS or disk.
No secrets: complete assembly language source code and tutorial provided

Quantasm Power Lib

- Over 400 fast, compact assembly language routines
- Overlapping windows in less than 3K
- Over 100 string and numeric functions
- Menus, date, time, sound and more!

Magic TSR Toolkit . . . $199.95
Power Lib. 99.95
Power Lib w/source . . 299.95
ASMFLOW. 99.95

Shipping & Handling: U.S. $5; Foreign $15
CA res. add sales tax
VISA and MasterCard accepted
30-day money back guarantee

QUANTASM CORPORATION

19855 Stevens Creek Blvd., Suite 154
Cupertino, CA 95014
408/244-6826

Circle **50** on Reader Service Card

FIGURE 3: General PS/2 BIOS map

E000	Copyright notice
	Int 5/Int 11h/Int 12h-General
	Int 0e,40h-Floppy support
	Int 13h,76h-Hard disk support
	Int 9,16h-Keyboard support
	Int 10h-Video support
	Int 15h-Extended support
	Advanced BIOS
	End-of-ROM chip filler
F000	Power-on-self-test
	Int 19h-Bootstrap support
	Int 02-NMI
	Int 1a-Real-time clock
	Int 17h-Printer support
	Int 14h-Serial support
	ROM chip filler
	BASIC

tediously disassembling two dozen BIOSes. As is the case with any program, the processing locations of specific BIOS functions can vary. The only genuine constant appears to be the last 8K starting at F000:E000. This informal BIOS entry point area provides a hard address for BIOS functions, thereby bypassing the interrupt jump table at 0:400. Because a small number of (mostly hardware-diagnostic) programs bypass the interrupt jump table, the entry point area is not compressed by BlueMAX, so that these programs can run properly.

By examining the BIOS entry points that are shown in Table 2 (p. 66), you can see why the entry point area is informal: several functions are missing, such as INT 13h, 76h, 41h, 46h, etc.

Using the above techniques, Qualitas is able to free up as much as 144K of memory for DOS. BIOS compression is one of those rare approaches that, unlike solutions based on smoke and mirrors, works every time.

Break Through
The 640k Barrier

with Memory
Management Libraries
fromTHE TOOL MAKERS

PowerSTOR®
Configure Expanded Memory, Extended Memory or Hard Disk as virtual memory. Allocations from 1 byte to 1 gigabyte. Ideal for data array structures. Supports most standard C compilers.

The Heap Expander®
Access Expanded Memory or Extended Memory. Swaps to Hard Disk as needed. Ideal for link list data structures. Supports Microsoft C and Turbo C.

Call Toll Free to order
USA
800-248-1045 ext 100

For more information, call The Tool Makers direct.

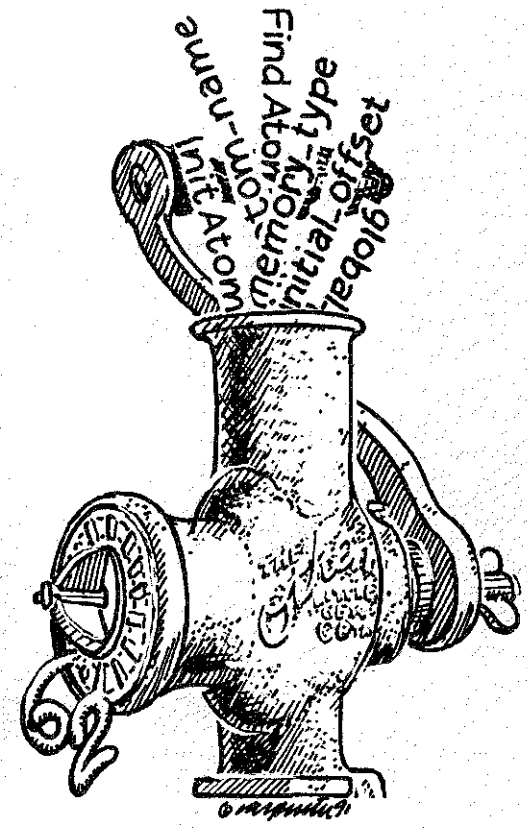
THE TOOL MAKERS
P.O. Box 2151
Santa Cruz, CA 95063
408-458-0690

Circle **49** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Hashing functions

Handle-based dynamic storage and hashing routines for strings



By John Otken

This article discusses assembly language facsimiles of two groups of functions from the Microsoft Windows Software Development Kit (SDK). The first is a handle-based dynamic storage system, and the second is a set of hashing routines for strings.

The first step in this project is to design the application interface (API). An API specifies procedure names and calling conventions.

Procedure names come from corresponding Windows functions. To avoid any possible conflicts with Windows names, we can transform procedure names to lowercase and add underscore characters between words in the function names. **AddAtom,** for example, becomes **add_atom**.

Because the corresponding SDK functions have no more than three parameters, adapting the calling conventions to the 80x86 register set is simple; optimal calling conventions for 80x86 procedures pass numeric values in AX and DX, handles in BX, counts in CX, input pointers in DS:SI, and output pointers in ES:DI. All of the "borrowed" SDK functions fit cleanly into this model. (For more information on register usage, see *PJ* 7.5.)

HANDLE-BASED STORAGE

A previous article (*PJ* 9.1) discusses the concept of 16-bit "handles" to identify storage blocks. In that article, storage blocks are a fixed length—16K. The fixed length eases swapping blocks to disk or EMS, but it lacks flexibility.

The "global" storage system, however, supports buffer sizes from 0x0010 to 0xFFF0 bytes (see Listing 1). To allocate a buffer, call **global_alloc** with a byte count in CX. **global_alloc** returns the actual buffer size in AX and a nonzero handle in BX. When the carry flag is on, there is insufficient memory; then the program should ignore values in AX and BX.

Under Windows, **GlobalAlloc** supports several different types of memory, including moveable, discardable, and fixed. Here, **global_alloc** supports only moveable memory. To allocate memory initialized to zero, call **global_calloc**.

Call **global_lock** with the buffer handle in BX to access a buffer. **global_lock** increments the lock count for the buffer and returns a pointer in DS:SI. While the lock count is nonzero, the buffer cannot move.

When your program completes buffer access, call **global_unlock** to decrement the lock count. Notice that **global_unlock** does not modify AX or the flags. When the lock count goes to zero, **global_unlock** will zero the segment registers (DS or ES) that point to the buffer. This ensures compatibility with the protected mode. Currently, **global_unlock** does not modify any segment registers.

BUFFER MANAGEMENT

Unlocking buffers can be difficult to program, but it is the safest way for the storage system to reduce memory fragmentation or to swap buffers to secondary storage. If a storage system assumes that a program has no pointers to a buffer on the basis of "least recently used," the memory system cannot guarantee against program failure that can occur when memory is in very short supply.

Changing the size of a buffer is easy in a handle-based storage system. Call **global_realloc** with a handle in BX and with the new size in CX. If the system cannot change the size of the block in place, it allocates a new block and copies the contents. Because locked buffers cannot move, avoid increasing their size.

To discard memory in a buffer and return the handle, call **global_free** with the buffer handle in BX. To ease programming, **global_free** ignores a

John Otken is president of Soft Advances, where he hacks assembly language into debuggers and develops other software for embedded systems. He can be contacted by writing to him at P.O. Box 184, Orinda, CA 94563.

ILLUSTRATION BY JIM CARPENTER

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LISTING 1

```
        title   global storage functions
        include asm.inc

        public  global_alloc, global_calloc, global_realloc, global_free,
        public  global_lock, global_unlock

GLOBAL_MAX              equ     100     ; maximum number of storage handles
NULL_HANDLE             equ     0

global_str struc
  g_address             dd  ?           ; storage address
  g_lock                dw  ?           ; storage lock count
  g_size                dw  ?           ; size of storage
global_str ends

        .data?
global_count            dw      ?       ; number of allocated handles
global_table    global_str      GLOBAL_MAX dup(<>)

        .code
        extn    calloc,clear_strerror,free,malloc,realloc

;;      global alloc
;       entry   CX      requested size
;       exit    AX      actual size
;               BX      storage handle (never zero)
;               Cf      if no storage
global_alloc proc
        pushm   di,es
        call    malloc
        jc      gal1                    ;  if not enough memory
        call    global_alloc_common
gal1:   popm    es,di
        ret
global_alloc endp

;;      global alloc common
;       entry   AX      storage size
;               ES:DI   storage pointer
;       exit    BX      storage handle
;               Cf      if out of handles
global_alloc_common proc
        pushm   ax,dx,cx,si,ds
        mov     dx,ax

        mov     bx,GLOBAL_MAX
        mov     cx,bx
        cmp     cx,global_count[bp]
        jbe     gac2                    ;  if out of handles

        mov     si, @data
        mov     ds,si
        lea     si,global_table-size global_str
gac1:   add     si,size global_str      ; search for first free handle
        mov     ax,wptr g_address[si]   ;   (SLOW!)
        or      ax,wptr g_address[si+2]
        loopnz  gac1
        jnz     gac2                    ;  if unexpected error: corrupted
                                        ;    data

        mov     g_lock[si],ax
        mov     g_size[si],dx
        mov     wptr g_address[si],di
        mov     wptr g_address[si+2],es
        inc     global_count[bp]

        sub     bx,cx                   ; compute non-zero handle 1..n
gac2:   popm    ds,si,dx,cx,ax          ;   (Cf=0)
        ret

gac3:   stc
        jmp     gac2
global_alloc_common endp

;;      global calloc
;       entry   CX      requested size
;       exit    AX      actual size
;               BX      storage handle (never zero)
;               Cf      if no storage
global_calloc proc
        pushm   di,es
        call    calloc
        jc      gca1                    ;  if not enough memory
        call    global_alloc_common
gca1:   popm    es,di
        ret
global_calloc endp

;;      global free
;       entry   BX      storage handle (OK if zero)
;       exit    BX      0
;       uses    AX
global_free proc
        pushm   di,es
        cmpx    bx,NULL_HANDLE
        je      gfr1                    ;  if NULL handle, just return
        call    read_global_entry
        mov     bx,NULL_HANDLE
        jc      gfr1                    ;  if bad handle

        les     di,g_address[si]
        call    free
        mov     wptr g_address[si],di
        mov     wptr g_address[si+2],es

gfr1:   popm    es,di
        ret
global_free endp

;;      global lock
;       entry   BX      valid storage handle
;       exit    DS:SI   storage pointer
;       note            DOES NOT BASH AX
global_lock proc
        call    read_global_entry
        jc      glk1                    ;  if bad handle
        inc     g_lock[si]
        lds     si,g_address[si]
glk1:   ret
global_lock endp

;;      global realloc
;       entry   BX      storage handle
;               CX      new size
;       exit    AX      actual size (if no errors)
;               Cf      if not enough memory
global_realloc proc
        pushm   di,si,ds,es
        call    read_global_entry
        jc      gre1                    ;  if bad handle

        les     di,g_address[si]
        call    realloc
        jnc     gre1                    ;  if realloc OK

        mov     ax,g_lock[si]           ; check lock count before moving
        add     ax,-1
        jc      gre1                    ;  if locked, cannot move
        call    clear_strerror          ;  else clear realloc error

        call    malloc
        jc      gre1                    ;  if no memory

        pushm   ax,di,es                ; copy old storage to new storage
        pushm   cx,si,ds
        mov     cx,g_size[si]
        lds     si,g_address[si]
        rep     movsb
        popm    ds,si,cx

        les     di,g_address[si]        ; free old storage
        call    free
        popm    es,di,ax

        mov     wptr g_address[si],di   ; set new storage pointer and size
        mov     wptr g_address[si+2],es
        mov     g_size[si],ax
        clc

gre1:   popm    es,ds,si,di
        ret
global_realloc endp

;;      global unlock
;       entry   BX      valid storage handle
;       note            flags do NOT change
global_unlock proc
        pushf
        pushm   si,ds
        call    read_global_entry
        jc      gun1                    ;  if bad handle
        dec     g_lock[si]
gun1:   popm    ds,si
        popf
        ret
global_unlock endp

;;      read global entry
;       entry   BX      global handle
;       exit    DS:SI   global table for handle
;               Cf      if bad handle
;
read_global_entry proc
        pushm   ax,dx
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

zero handle in BX. Programs can free buffers without checking for a NULL handle.

This version of the global storage system is immature in several respects. It lacks error checking. If a program calls a storage procedure that has a bad handle, the program is broken and should abort with a fatal error message.

The storage system does not migrate or swap buffers when free memory resources are low. It lacks a "shaker" mode that moves a buffer each time the buffer's lock count reaches zero. I learned the hard way that a storage shaker is critical for debugging handle-based storage systems. Fortunately, these features can be added without changing the interface. An enhanced version is planned for a future article.

SYMBOL HASHING

Hashing is a technique for building and searching symbol tables. It works by scattering symbol entries across a table and limiting searches to a few entries. Hashing is like using a small-town phone book: find the page for the first letter of the last name you're looking for and scan the names until you find it.

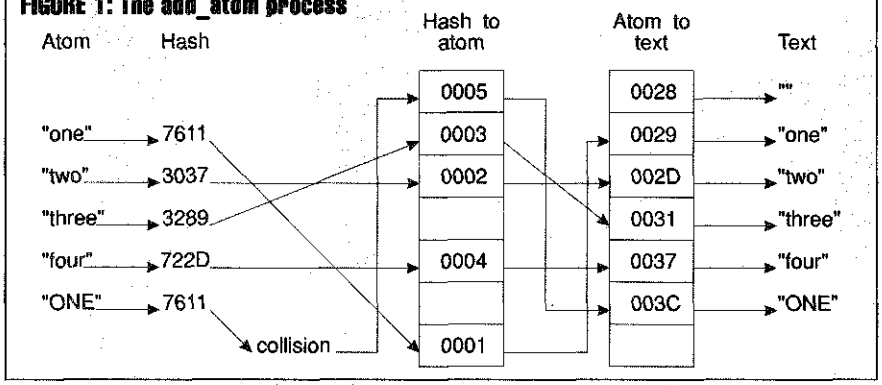
Unlike a phone book, however, hashing algorithms do not use alphabetical order. The hash function in the sample program combines exclusive **or**, **add**, and **rotate** instructions to jumble a symbol into a 16-bit value. The symbol's designated table entry is the remainder after dividing this value by the table size. The table size should be a prime number.

MIND YOUR TABLE MANNERS

To add a symbol to the table, use the hash function to find a table position. Search consecutive table entries for a zero or a matching symbol. If the matching entry is zero, put the symbol in that position. Return the entry index or some other unique identifier for the symbol.

Searching for a symbol is similar to adding one. Use the hash function to find a table position. Search consecutive table entries for a zero or a matching symbol. If the matching entry is zero, the symbol is not in the table; otherwise, return an identifier.

FIGURE 1: The add_atom process



Because the hash function can return the same value for different symbols, the hash value cannot be used to identify the symbol and is not stored in any tables. Use the hash value itself only to find where to start searching.

A simple hashing system has one table with pointers to symbol text. It identifies symbols by their table index. For some programs, the "random" output of this system is a problem. A more orderly system uses two tables. The hash table holds symbol identifiers, and a second table holds pointers to symbol text. This latter system uses more memory but returns consecutive identifiers for new symbols.

Unlike your car's gas tank, a hash table should never be full. In fact, the algorithm I've used here breaks when there are zero free-table entries. A nearly full hash table is also bad. It performs as slowly as a linear search—one that has just lost its mind. A hash table should be no more than half full.

Hashing symbol tables offers the assembly language programmer more than speed. Because hashed symbols reduce symbols to a number, they are much easier to manipulate than strings. Moving a 16-bit identifier is much easier than moving a 32-bit pointer or a variable-length string. An identifier can reference multimegabyte symbol spaces (swapped to EMS), whereas a real-mode pointer cannot. Comparing hashed symbols is trivial; if the identifiers are equal, the symbols are equal.

UP AND ATOM

Windows uses the keyword "atom" in the names of its hashing routines. The functions adapted for the sample program (see Listing 2) are **AddAtom**, **FindAtom**, **GetAtomName**, and **InitAtomTable**.

Before using other atom routines, call **init_atom_table** with the table size in AX. This number should be at least double the expected number of symbols. It should also be a prime number; do not use a power of two.

To keep the code simple, all data structures fit in one 64K buffer; strings are stored in the same buffer. This limits the total number of symbols to about 4K. The prime number closest to 4,096 is 4,093.

To add a symbol to the table, call **add_atom** with a pointer to the null-terminated string in DS:SI. **add_atom** returns AX with identifying "atoms" starting with one. Each new symbol returns the next consecutive number. Symbols already in the table return their previously assigned number. When the table fills, **add_atom** sets the carry flag.

AddAtom under Windows uses atoms from 0xC000 to 0xFFFF; the sample program uses atoms from 1 to 0xFFFF. This should make no difference to most programs. The zero atom is reserved for the zero-length symbol. Do not confuse this with a NULL pointer; calling **add_atom** with DS:SI equal to zero is a programming error.

In order to search for a symbol, call **find_atom** with a string pointer in DS:SI. **find_atom** either returns the corresponding atom in AX or sets the carry flag if the symbol isn't found.

To convert an atom into a string, call **get_atom_name** with an atom in AX, an output pointer in ES:DI, and a maximum string length in CX. **Get_atom_name** copies the atom's string and updates CX and DI.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ADVERTISER INDEX

Reader Service	Advertiser Name	Page Number
38	Austin Code Works, The	50
37	Autumn Hill Software	49
110	Az-Tech Software, Inc.	85
40	Blaise Computing, Inc.	55
43	Block Island Technologies	59
56	Borland International	C4
113	Burton Systems Software	85
118	C.C. Software	85
17	Catenary Systems	20
57	Cater Software	19
*	CDP Consultants	87
*	C Gazette	29
134	Cnapse	87
135	Device Drivers, Inc.	87
21	Disk Software, Inc.	22
137	DSE Software Publishing	85
111	EZX	85
5	Genus Microprogramming, Inc.	8
*	Gimpel Software	53, 86
25	Grammar Engine, Inc.	30
12	Greenleaf Software, Inc.	15
*	GUI Computer	25
115	Impulse Engineering	85
130	Information Modes	87
26	Innovative Data Concepts	30
123	Insync Software	86
112	Interstitial Software Systems	85
27	Kadak Products, Ltd.	31
45	Kandu, Inc.	67
8	Knowledge Dynamics Corp.	10
48	Laboratory Microsystems, Inc.	67
35	Lahey Computer Systems, Inc.	48
19	Magee Enterprises, Inc.	20
*	Mark Williams Company	13
3	MetaWare Incorporated	5
121	MicroQuill	86
47	Micro Security Systems	25
*	Microway	39
28	Mix Software	35
13	Modular Software Systems	17
24	Nanosoft	28
117	Network Integrated Services	85
2	Nu-Mega Technologies	2
*	Olivadoti Associates, Inc.	85
120	Omni-Tech Software	86
127	Opt-Tech	87
9	Opus Software	10
132	Orion Software	87
18	PC Tech, Inc.	20
44	Periscope Company, Inc., The	59
52	Petrasoft	37
*	***PJ* Mail Order Store**	73
11	Programmer's Corner, The	12
15	ProGraphx Toolbox	18
137	Quaid Software Limited	86
41	Qualitas, Inc.	56
50	Quantasm Corporation	68
1	Quarterdeck Office Systems	C2
55	Quarterdeck Office Systems	C3
32	Rational Systems, Inc.	43
6	RSS Engineering	9
4	RJSwantek & Associates	7
54	RR Software, Inc.	82
129	Ryle Design	87
16	S.L.A.M. Software	19
136	Selective Software	87
42	Sequiter Software, Inc.	57
29	Softelligence	35
116	Software Mart	85
131	Software Success	87
122	Software Toolz, Inc.	86
23	Softway, Inc.	23
53	Solution Systems	79
14	Sub Systems	18
125	Sutrasoft	86
22	Symmetry Group, The	23
49	Tool Makers, The	68
10	Tools & Techniques, Inc.	12
128	TSR Systems	87
30	Turbopower Software	37
20	Turbo User Group	22
133	Upper Deck Systems	87
7	Utilis Inc.	10
51	V Communications	74
*	Vermont Database Corp.	85
*	Watcom Products, Inc.	1
114	Wenham Software Co.	85
36	Western Wares	48
34	Willies' Computer Software	45
33	Winsoft	45
39	Zircel Software	52

When contacting any of the above advertisers, please tell them you saw their ad in ***Programmer's Journal*** and that you appreciate their continuing support of this magazine. The advertisers marked with an asterisk (*) prefer to be contacted directly. ***PJ***'s Advertiser Index is provided as a free service, and ***Programmer's Journal*** cannot be held responsible for omissions or errors. ***Programmer's Journal*** is not responsible for errors in advertisements.

PJ Code is available on the following bulletin board services:

Buffer Board
538 Cardinal Way
Springfield, OR 97477
(Data) 503-747-7636
The Buffer Board contains all of PJ's code from Sept/Oct 1987 issue through the current issue.

PCS-49 on CompuServe
Section 10
Sponsored by:
MISOSYS, Inc.
P.O. Box 239
Sterling, VA 22170
(Voice) 703-450-4181

Gilmore Systems of Southern California
5447 Indian Hills Drive
Simi Valley, CA 93063
(Voice) 805-582-1360
(Data) 805-582-9306

Generation Five Inc.
12061 Tech Road
Silver Spring, MD 20904-9998
(Voice) 301-680-3600
(Data) 301-680-9300
Accessible via PC Pursuit, Washington, D.C.
From Files submenu, select "P" for Personal Downloads.

A.C.M. Phoenix Chapter
6408 East Palm Lane
Scottsdale, AZ 85257-2563
(Data) 602-970-0474
(Voice) 602-990-3102

Mnematics Videotex
P.O. Box 19
Sparkill, NY 10976-0019
(Voice) 800-322-3633
(Voice) 914-365-0184 in N.Y.
*Mention **PJ** when you call.*

Autumn Hill Software Inc.
1145 Ithaca
Boulder, CO 80303
(Voice) 303-494-8865
(Data) 303-494-8868

The Programmer's Corner
5430F Lynx Ln. #224
Columbia, MD 21044
(Data) 301-596-1180, 621-4636, or 995-3744

The Silicon Zone
4419 W. Hillsboro Blvd.
Suite 290
Coconut Creek, FL 33073
(Data) 305-360-9376

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

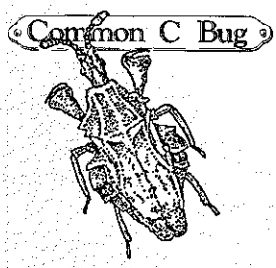
Multi-color imprints on the highest quality 100% cotton T-shirts.

SIZES: S, M, L, XL, XXL

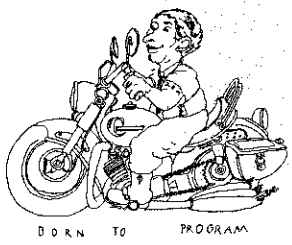
$14.95 each — includes U.S. shipping

PJ ISSUE SPECIAL!

SAVE $10.00 ON ABRASH BOOK & DISK



COMMON C BUG



BORN TO PROGRAM

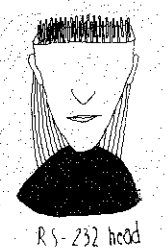
Power Graphics Programming

ABRASH BOOK AND CODE DISK

For a limited time, you can save over 25% when you order both the *Power Graphics Programming* book and the companion code disk during the ***PJ*** 9.3 Issue Special. Lowest price ever! Order your book and disk today for only $29.90 and save $10.00.

(Offer expires June 30, 1991. All orders must be prepaid.)

PARALLEL PORTS



RS-232 HEAD

BUGS RULE 4 color imprint on a white shirt

$7.00 each PREPAID ONLY

A few of the Back Issues currently available

Vol. 8.1: NETWORKING: Network File System; an overview of NetBIOS on the PC, from *C Programmer's Guide to NetBIOS* by W. David Schwaderer; fast circle drawing on the VGA; Dynamic Data Exchange in Windows; DESQview API features; computing time in C; a menu system with Turbo Pascal 5.5.

Vol. 8.2: GRAPHICS: Faster circles for the VGA; the TIGA interface from TI; the EGA palette; 800x600x16 Super-VGA mode programming; dynamic data exchange for communications; Manipulating large data; fast file I/O in QuickBASIC; industry news; 1989 article index.

Vol. 8.3: OPERATING ENVIRONMENTS: Undocumented DOS shell; graphics programming in windows; mousing around: messages in PM; accessing high memory; ASM:advanced segmentation; ellipses on the VGA; Up OOP alley in Pascal.

Vol. 8.4: PROGRAMMING IN THE PROTECTED MODE: Faster VGA ellipses and more; Running unprotected virtual 8086 mode; Converting to 386 protected mode; OS/2 2.0 memory model for the 386; Fortran for the 386; Using C++ features in C; Different times compensating for system clock idiosyncracies; ASM:Errors and Stream I/O; Cobol-85 nested; Deliberately breaking the rules.

Vol. 8.5: OBJECT_ORIENTED PROGRAMMING: More colors on the VGA; A virtual memory system in C/C++; Pitfalls in Turbo Pascal OOP; Writing applications that travel; Late binding; Tools for 386 DOS; Windowing text with X; Cache memories; Breaking the rules again; SPARCs plot world takeover.

Vol. 8.6: THE DATABASE CHALLENGE: VGA color paging; The database dilemma; Networking databases; B-tree indexing; Virtual memory paging; Writing an ANSI C preprocessor; Peak performance: on to the 80486; Calendar conversions; Software components; Optimization strategies; Pink noise; Oh say can you C?

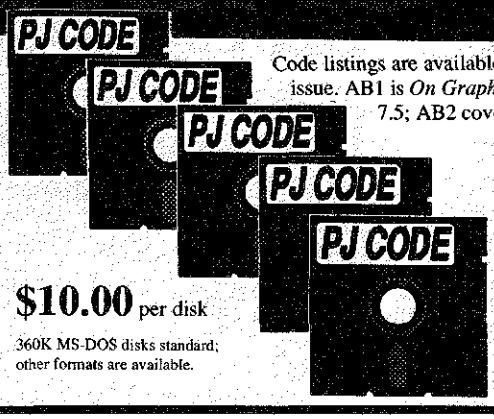
Vol. 9.1: GRAPHICS AND WINDOWING: VGA color cycling; Digital typography; Windowing for two; GUI madness; Text buffers in windows; What's on the menu; 386 DOS meets ANSI C; Optimania; Real-mode virtual memory; Enhanced keyboards; An open look at Motif; Negotiating tricks and traps; For a few good books; Goliath beats David—As usual.

Vol. 9.2: LANGUAGES: Windows 3 input hooks; C compilers for Windows; Fortran 90 is coming!; Pragma interface; String handling; Porting to the Mac; Disassembler roundup; Exploring XMS; Using EMA for fast file I/O; Around the bend again; Pink noise; Language lore; Effective marketing presentations; COBOL revisited; Language parsers and Fortran tools; Changes in the fast lane.

10% discount on any order of 10 or more T-Shirts, Disks or Back Issues.

All mail orders must be accompanied by check or Money Order -- PREPAID ONLY.

Use the order form on the card opposite.
Order by phone with your VISA/MC.
CALL TOLL FREE
800/234-0386 ext 53



$10.00 per disk

360K MS-DOS disks standard; other formats are available.

Code listings are available from volume 5.5 to the current issue. AB1 is *On Graphics* series code from issue 5.1 to 7.5; AB2 covers issue 7.6 to the current issue.

Diskettes are available for the following code:

AB1	6.4	7.5	8.6
AB2	6.5	7.6	9.1
5.5	6.6	8.1	9.2
5.6	7.1	8.2	9.3
6.1	7.2	8.3	
6.2	7.3	8.4	
6.3	7.4	8.5	

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

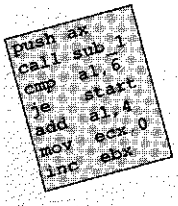
Some Assembly Required!

V Communications has all the tools you need to develop, examine, verify, and document assembly-language programs, quicker and easier than ever before.

New!

ASM Checker™ - Locates Coding Problems

ASM Checker automatically locates common coding errors that are often hard to find and easily overlooked. It also cross-references external and public symbols and identifies unused symbols and type mismatches. It can save you countless hours of debugging time. $179.95



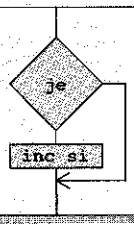
Sourcer 486™ - Commenting Disassembler

Sourcer creates commented source code and listings from memory or executable files. It helps you understand the inner details of how programs work. Sourcer output is suitable for reassembly, so you can easily make changes or correct problems in programs. $129.95

"Sourcer is the best disassembler we've ever seen!"
- PC Magazine

BIOS Pre-Processor™ - BIOS Listings

The BIOS Pre-Processor lets you use Sourcer to obtain commented listings for any BIOS ROM in your machine. Now you can understand how your specific BIOS works. Adds over 75K of comments specific to your BIOS. $49.95
Sourcer with BIOS Pre-Processor $169.95 (Save $10)



ASMtool 486™ - Flowcharting and Analysis

ASMtool makes complex assembly source easy to understand. It automatically creates flowcharts and procedure call diagrams that clarify logic paths. It also provides static timing analysis for each procedure, register usage analysis, and more. $199.95

More Performance Tools from V Communications

View-It™ - Fast, zoomable file viewer, easily view wide 132-column documents and assembly listings.	$69.95
Unpacker™ - Unpacks packed EXE files and more!	$39.95
ASM ProPak III™ - Sourcer 486 with BIOS, ASMtool 486, Unpacker, View-It, and ASM Checker. (Save $120)	$549.70
Quantasm PowerLib™ - Library of over 300 documented assembly routines with source. (Object only: $89.95)	$279.95
Quantasm Magic TSR Toolkit™ - Build your own TSRs. Documented assembly source code and examples.	$199.95
C-Clearly® - Formats C source code your way.	$129.95
Memory Commander™ - Complete 386/486 memory manager that provides up to 900K of main DOS memory.	$99.95

Specifications: Works on all PC and PC-compatible computers with DOS 2.0 or later. Assembly products support 8088 through 80486 and math instruction sets and are designed to work with Microsoft MASM or Borland TASM assemblers. View-It requires EGA/VGA/XGA for zoom features. All product names are trademarks of their respective companies.

Shipping and Handling: USA $6; Canada/Mexico $10; Other $18. CA residents add sales tax. VISA/MC/AMEX.

Call 1-800-648-8266

V Communications, Inc.

4320 Stevens Creek Blvd Suite 275 Dept. PT9 San Jose CA 95129 fax 408.296.4441 408.296.4224

Circle 51 on Reader Service Card

TABLE 1: Memory layout

Addr	Contents	Description
Atom variables		
+0000	0006	atom_count
0002	0007	hash_divisor
0004	000C	hash_to_atom
0006	001A	atom_to_text
0008	0040	end_of_text
000A	0430	storage_size
Hash to atom table		
+000C	0005	"ONE" hashes here (after collision)
000E	0003	"three" hashes here
0010	0002	"two" hashes here
0012	0000	
0014	0004	"four" hashes here
0016	0000	
0018	0001	"one" hashes here
Atom to text table		
+001A	0028	points to ""
001C	0029	points to "one"
001E	002D	points to "two"
0020	0031	points to "three"
0022	0037	points to "four"
0024	003C	points to "ONE"
0026	0000	
Text table		
+0028	\0	
0029	one\0	
002D	two\0	
0031	three\0	
0037	four\0	
003C	ONE\0	

The sample code (see Listing 3) creates an atom table that has five symbols: "one," "two," "three," "four," and "ONE." Table 1 shows the memory layout after the symbols have been added. The atom table has four parts. The first part holds some variables for the hash system. The second part has the scatter table that maps values from the hash function to atom numbers. The third is the pointer table that maps atom numbers to text pointers. The text table is last, so it can expand as symbols fill available space. The scatter and pointer tables do not expand.

Figure 1 (p. 71) follows the **add_atom** process. The "Hash" column shows a raw hash for each symbol. The next column shows the position of these symbols in the scatter table after dividing the raw hash by the table size. The "Atom to text" column shows the pointer table, and the last column shows the text table.

The actual algorithm does not follow exactly the left-to-right flow of

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
        mov     ax, @data
        mov     ds,ax
        lea     si,global_table

        cmp     bx,GLOBAL_MAX
        ja      rgel                    ;  if bad handle

        mov     ax,bx                   ; multiply handle by structure size
        dec     ax                      ;  to select global table entry
        mov     dx,size global_str
        mul     dx
        add     si,ax
rgel:   popm    dx,ax
        ret
read_global_entry endp

        end
```

LISTING 2

```
        title   atom management routines
        include asm.inc

        public  add_atom                ; add string atom to hash table
        public  add_binary_atom         ; add binary atom to hash table
        public  find_atom               ; search for string atom
        public  find_binary_atom        ; search for binary atom
        public  get_atom_name           ; copy atom text
        public  init_atom_table         ; initialize atom table

atom_str struc
  at_atom_count         dw  ?           ; number of atoms (including NULL)
  at_hash_divisor       dw  ?           ; prime divisor (table entry max)
  at_hash_to_atom       dw  ?           ; hash offset to atom index table
  at_atom_to_text       dw  ?           ; index offset to symbol text table
  at_end_of_text        dw  ?           ; first free byte in symbol table
  at_storage_size       dw  ?           ; dynamic storage byte count
atom_str ends

        .data?
atom_table              dw      ?       ; storage handle for hash table

        .const
ertx_bad_atom_index     db      'Bad atom index',0
ertx_atom_table_full    db      'Atom table full',0
ertx_atom_too_big       db      'Atom too big',0

        .code
        extn    global_calloc,global_free,global_lock,global_realloc
        extn    global_unlock,save_most,strlen,strncpy,set_strerror
;;      add atom
;       entry   DS:SI   string to hash
;       exit    AX      atom index
;               Cf      if table full
add_atom proc
        push    cx                      ; add null terminated string to
        call    strlen                  ;  atom table
        call    add_binary_atom
        pop     cx
        ret
add_atom endp

;;      add binary atom
;       entry   DS:SI   bytes to hash
;               CX      byte count (<256)
;       exit    AX      atom index
;               Cf      if table full or atom too big
add_binary_atom proc
        call    find_binary_atom
        jnc     aba5                    ;  if duplicate atom

        or      ch,ch
        jnz     aba6                    ;  if atom is too long

        call    save_most
        mov     ax,ds                   ; use ES:DI as atom pointer
        mov     es,ax
        mov     di,si
        call    compute_atom_hash

        mov     bx,atom_table[bp]       ; get pointer to atom table
        call    global_lock

        mov     ax,at_atom_count[si]    ; check atom count
        inc     ax
        cmp     ax,at_hash_divisor[si]
        jae     aba7                    ;  if too many symbols

        mov     ax,at_storage_size[si]  ; check empty text space
        sub     ax,at_end_of_text[si]
        cmp     cx,ax
        jae     aba8                    ;  if no space for new atom text
aba1:   mov     ax,at_end_of_text[si]   ; set text pointer in index table
        mov     bx,at_atom_count[si]
        add     bx,bx
        add     bx,at_atom_to_text[si]
        mov     [si+bx],ax

        mov     bx,ax                   ; copy atom text
aba2:   mov     al,es:[di]
        mov     [si+bx],al
        inc     di
        inc     bx
        loop    aba2
        mov     [si+bx],cl              ;  (delimit atom text \0)
        inc     bx
        mov     at_end_of_text[si],bx   ;  update end of text pointer

        mov     cx,dx                   ; find free entry in hash table
aba3:   mov     ax,cx
        inc     cx                      ;  (bump hash for possible
                                        ;   collision)
        mov     dx,ZERO
        div     at_hash_divisor[si]
        mov     bx,dx
        add     bx,bx
        add     bx,at_hash_to_atom[si]
        cmp     word ptr [si+bx],ZERO   ;  (Cf=0)
        jnz     aba3                    ;  if hash table collision

        mov     ax,at_atom_count[si]    ; set atom index in hash table
        mov     [si+bx],ax
        inc     ax                      ; advance atom count
        mov     at_atom_count[si],ax
        dec     ax                      ;  (Cf cleared above)
aba4:   mov     bx,atom_table[bp]       ; unlock atom table
        call    global_unlock
aba5:   ret

aba6:   lea     ax,ertx_atom_too_big    ; *Atom too big*
        call    set_strerror
        jmp     aba5

aba7:   lea     ax,ertx_atom_table_full ; *Atom table full*
        call    set_strerror
        jmp     aba4

aba8:   mov     ax,at_storage_size[si]  ; increase size of atom table
        add     ax,1024                 ;  (by 1k)
        jc      aba7                    ;  if storage would exceed 64k

        push    cx
        mov     cx,ax
        call    global_unlock
        call    global_realloc
        pop     cx
        jc      aba5                    ;  if realloc failed
        call    global_lock
        mov     at_storage_size[si],ax
        jmp     aba1
add_binary_atom endp

;;      compute atom hash
;       entry   DS:SI   atom pointer
;               CX      byte count
;       exit    DX      case insensitive hash
;       uses    SI
compute_atom_hash proc
        push    cx
        xor     dx,dx
        jcxz    cah2                    ; if null atom
cah1:   xor     dl,[si]
        inc     si
        or      dl,20h                  ; (make hash case insensitive)
        add     dx,cx
        rol     dx,1
        rol     dx,1
        rol     dx,1
        rol     dx,1
        loop    cah1
        xor     dl,dh
cah2:   pop     cx
        ret
compute_atom_hash endp

;;      find atom
;       entry   DS:SI   string to find
;       exit    AX      atom index
;               Cf      if not found
find_atom proc
        push    cx                      ; find null terminated string in
        call    strlen                  ;  atom table
        call    find_binary_atom
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

this figure, but it will help you understand the code. Examine the figures and try mentally tracing the code. If it makes sense, load the program and single-step the code with a debugger.

OTHER ALGORITHMS

There are many different hash functions for processing symbols. You can find or create functions that are faster or "tuned" for a particular set of keywords.

Many different data structures can represent hash tables and different algorithms for "conflict resolution." The sample program's "open addressing" method is simple, but it isn't suitable for deleting table entries. A slightly more complex algorithm that supports deletion is called "chaining." When symbols collide under an open addressing scheme, the new symbol goes into the first available entry after the point of impact. Under a chaining scheme, symbols form a linked list from the scatter table. I found open addressing to be less complicated than chaining.

A technical bookstore should have references that describe several different hashing algorithms. I use Volume 3 ("Sorting and Searching") of Donald Knuth's *The Art of Computer Programming* (Addison-Wesley, 1973). PJ

```
        pop     cx
        ret
find_atom endp

;;      find binary atom
;       entry   DS:SI   bytes to hash
;               CX      byte count (<256)
;       exit    AX      atom index (or 0)
;               Cf      if not found
find_binary_atom proc
        xor     ax,ax                   ; zero is null symbol index (Cf=0)
        jcxz    fba3                    ;  if null symbol
        or      ch,ch
        jnz     fba4                    ;  if atom too big

        pushm   bx,dx,di,es
        mov     ax,ds                   ; ES:DI is target symbol text
        mov     es,ax
        mov     di,si
        call    compute_atom_hash
        inc     cx                      ;   (include \0 in compares)

        mov     bx,atom_table[bp]       ; get pointer to atom table
        call    global_lock
        mov     bx,si

fba1:   xchg    ax,dx                   ; divide hash by table size
        mov     si,ax                   ;   ( a prime #)
        xor     dx,dx
        div     at_hash_divisor[bx]
        xchg    dx,si                   ; use remainder as hash table index
        inc     dx                      ;  (advance to next hash entry)
        add     si,si                   ;  get atom index from hash table
        add     si,bx
        add     si,at_hash_to_atom[bx]

        lodsw
        cmpx    ax,0
        stc
        je      fba2                    ;  if entry empty, symbol not found

        mov     si,at_atom_to_text[bx]  ; translate index in AX to
        add     si,ax                   ;   displacement from beginning of
        add     si,ax                   ;   symbol text table
        mov     si,[bx+si]
        add     si,bx

        pushm   cx,di                   ; compare target symbol & table
                                        ;  entry
        repe    cmpsb                   ;  (CX includes \0 delimiter)
        popm    di,cx
        jne     fba1                    ;  if no match, try next table
                                        ;   entry

fba2:   mov     bx,atom_table[bp]
        call    global_unlock

        dec     cx                      ;  (restore CX adjusted for \0)
        mov     dx,es
        mov     ds,dx
        mov     si,di
        popm    es,di,dx,bx
fba3:   ret

fba4:   lea     ax,ertx_atom_too_big    ; *Atom too big*
        jmp     set_strerror
find_binary_atom endp

;;      get atom name
;       entry   AX      atom index
;               CX      buffer size
;               ES:DI   output buffer
;       exit    DI      updated (points to \0)
;               CX      updated
;               Cf      if bad atom index
;       uses    AX
get_atom_name proc
        pushm   bx,si,ds
        mov     bx,atom_table[bp]       ; access atom table
        call    global_lock

        cmp     ax,at_atom_count[si]    ; check atom index
        jae     gan1                    ;  if too high - bad index

        add     ax,ax                   ; use atom index to find atom text
        mov     bx,ax
        add     bx,at_atom_to_text[si]
        mov     bx,[si+bx]
        mov     si,bx

        call    strncpy                 ; write atom to output buffer
        clc
        jmp     gan2

gan1:   lea     ax,ertx_bad_atom_index  ; *bad atom index*
        call    set_strerror

gan2:   mov     bx,atom_table[bp]       ; unlock atom table (Cf unchanged)
        call    global_unlock
        popm    ds,si,bx
        ret
get_atom_name endp

;;      init atom table
;       entry   AX      prime number approximate twice maximum atom count
;       exit    Cf      if no memory
;       uses    AX
init_atom_table proc
        call    save_most
        mov     di,ax                   ; compute table size
        mov     dx,4                    ;  (space for atom_to_text and
        mul     dx                      ;   hash_to_atom tables)
        add     ax,size atom_str+1024   ;  (space for atom_str and
                                        ;   symbol_table)
        adc     dx,dx
        stc
        jnz     iat1                    ;  if table > 64k
        mov     cx,ax

        mov     bx,ZERO                 ; free old atom table
        xchg    bx,atom_table[bp]
        call    global_free             ;  (OK if BX==0)

        call    global_calloc           ; allocate and zero atom table
        jc      iat1                    ;  if not enough memory
        mov     atom_table[bp],bx

        call    global_lock
        mov     at_storage_size[si],ax
        mov     at_atom_count[si],1
        mov     at_hash_divisor[si],di
        mov     ax,size atom_str        ; hash_to_atom table follows
                                        ;  structure
        mov     at_hash_to_atom[si],ax
        add     di,di                   ; followed by atom_to_text table
        add     ax,di
        mov     at_atom_to_text[si],ax
        add     ax,di                   ; symbol table is last
        inc     ax
        mov     at_end_of_text[si],ax

        call    global_unlock
iat1:   ret
init_atom_table endp

        end
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Language translation and 386 debugging

By M. Steven Baker

This column started out with a simple goal: to test a number of symbolic debuggers and profilers for 386 DOS-extender development. As often happens, the project grew in the process, and my objectives had to be scaled down.

My first task was to choose some programs to debug. But what code do you use to test a debugger? It should be broken and need fixing. I wanted some decent-sized programs, so I decided to use two Fortran-to-C translators and convert to C some existing benchmark programs I've used before. These translators convert various dialects of Fortran to both 16- and 32-bit C source for either ANSI or older Unix-style K&R compilers. I also thought it would be interesting to compare the execution speed of translated C code with the original Fortran so that I could see what the new optimizing C compilers and ANSI C can deliver, particularly for numerics.

SOURCE CODE TRANSLATION

In *PJ* 9.2 I reviewed scanner and parser generator tools, so it seems natural to follow up with a look at products that could be developed using such programs. A wealth of PC tools are available for manipulating source code in various languages. This is fortunate, because most programmers somehow inherit older code from their predecessors. A programmer's quandary is whether to rewrite a program from scratch or work with existing code. If the amount of older code is large, there really isn't much choice. For example, I routinely work with large Fortran programs representing a hundred or more work-years of effort.

Source-code translation might be desired for several reasons. One might be moving to a new hardware environment that doesn't support the original source language. Moving older graphics code to TI's TMS340x0 graphics co-processor is a good example—C compilers are available but not much else. A policy decision may instigate a conversion to a different language. For example, one reason MetaWare changed from Pascal source compilers to C was to take advantage of the C training of computer science graduates. Part of a large program might be converted in order to take advantage of some specific feature available in another language (for example, to access C's X Window libraries).

FORTRAN-TO-C TRANSLATORS

Promula.FORTRAN, from OTG Systems, is a Fortran-to-C translator for the PC. The program comes with the translator (PFC), a dialect definition utility, and runtime library code. The translator supports a large set of command-line switches that control the translation process. The user has control over the Fortran source dialect and the target environment (for example, 16- or 32-bit ANSI or Unix C compilers). Options control COMMON block treatment, local storage allocation (static, auto, dynamic, or virtual), CHARACTER variable conversions, parameter passing, casting, various optimizations, I/O, C-style output format, and syntax error handling. PFC can write function prototypes or read a file of prototypes for use in the translation.

Promula is unique in that a dialect definition file determines much of the translation process. The source to the standard dialect definition file (DDF) is provided, along with a packing utility for use by the translator for other dialects. Different source and target dialects can be supported by using different DDF files. After becoming familiar with the DDF syntax, I was able to add support for several VAX Fortran extensions that Promula did not originally support.

Promula supplies C runtime library routines to simulate various Fortran operations, such as Fortran I/O and intrinsic routines (C has no equivalent to Fortran's complex data type). Object library files are provided for Microsoft, Turbo, and Lattice C. The runtime library source is provided along with batch files to rebuild the libraries. The translator and runtime library also support virtual memory management routines to virtualize large static data arrays often found in Fortran programs.

The runtime library is large (570K source in over 240 files), and it took me several days to build runtime libraries for the various memory models of DOS C and 386 DOS-extender C compilers. The manual (a large three-ring binder) comes in two parts—the translator manual and a Fortran language reference manual. The translator section covers using Promula.FORTRAN, the supported default dialect, the DDF syntax, and the runtime library.

The Promula translator does only limited expression and optimization, so even fixed-iteration Fortran DO loops are converted to C "for" loops with *longs* as indices. The virtual data option allows large programs to be ported to the PC, but at a substantial penalty in speed (large arrays become disk-based). The version I used could export function prototypes to a file but couldn't use these prototypes and generated only Unix-style C code. Promula had a hard time swallowing large Fortran modules. Nevertheless, I really liked this product, particularly its capability to add features using the dialect definition file. The translator performed well and created translations that, in general, compiled cleanly. The Promula staff were responsive, quickly providing bug fixes. The PC and Mac versions are $995, and the Unix workstation version is $1,495.

Cobalt Blue has released version 3.1 of their Fortran-to-C translator, **FOR_C.** The FOR_C package consists of a translator and the FOR_C runtime library. Object versions of the library for Microsoft and Turbo C are included. The source code for the runtime library with batch files to rebuild libraries is available at additional cost.

The FOR_C translator supports a large number of command-line options to control the translation process. These options can also be taken from a configuration file. Both 16- and 32-bit C environments (ANSI or Unix) can be targeted. Options support local data allocation (automatic, static, or dynamic), various COMMON translations (structures and externals), DO options, C-style I/O, expression optimizations, C output style

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

and format, Fortran CHARACTER and Hollerith data translations, and the generation of debugging code. FOR_C can create a file with C prototypes and read a prototype file before translation.

The translator reads a macro file at start-up that can rename variables, thus avoiding conflicts when translating to C. Macros can be used to modify argument lists or to pass through certain Fortran code constructs as literal lines or comments. FOR_C also supports a limited set of C-like preprocessing statements for Fortran. These macros and translator options provide some customization capability.

The FOR_C manual (a sizeable three-ring binder) covers the use of FOR_C, various Fortran-to-C translations, and runtime library routines. It does an excellent job of discussing conversion problems and illustrating the outcome of various command-line options on the generated C code. Appendices cover the Fortran statement syntax and the supported FORTRAN-77 intrinsics.

Although the FOR_C runtime library source is small (160K in 40 files), generating runtime libraries with it for several DOS and 386 DOS compilers still took a couple of days. FOR_C's runtime library is not granular; some files contain a number of routines (for example, many complex intrinsic routines in one module). The product performs only local optimizations—fixed-iteration DO loops are converted to *longs*. Although standard C I/O is supported, FOR_C generates C-style I/O only when the Fortran format statements occur *before* the actual Fortran READ and WRITE statements (i.e., a one-pass translation). Cobalt Blue's **FOR_STRUCT** Fortran restructuring utility (see "Impressions" in *PJ* 9.2) has an option to order format statements this way. FOR_C performed admirably. The C translations were generally complete, and most compiled cleanly. The PC version is $675 (for binaries), $975 (with runtime source); the Unix version is $1,250.

ON TO SOME TOUGH CODE

I took my usual Fortran benchmark programs and poured them through the translators. The result was a mixed bag of C code to work with—some of it good and some of it buggy. Although you can translate a program to C, don't expect DOS 16-bit C compilers to swallow the translation easily unless the Fortran program is a small one.

For larger programs, targeting 32-bit C compilers is more reasonable. The major obstacle is that most Fortran programs were written with large data arrays (statics) in COMMON blocks. Although PC Fortran compilers have adjusted to this and can transparently handle data objects larger than 64K, DOS C compilers provide only grudging support for *huge* data. Both Promula.FORTRAN and FOR_C have options to allocate "local" data dynamically above a certain user-defined threshold, but neither currently allocates global data (Fortran COMMON blocks). Lacking the Promula virtual data option, the only simple way to support static data for most translated Fortran programs is using the *huge* memory model of your favorite 16-bit C compiler. Although DOS C compilers support huge pointers in other memory models, large static data is supported only by using the huge memory model (MetaWare High C is the single exception). And of course, don't expect earth-shattering speed using the huge memory model.

The 386 DOS-extender C compilers should have no problems with a mere 64K of static data, right? Well, not exactly. Although MetaWare, MicroWay, and SVS C compilers will swallow large arrays and structures, Watcom C/386 (versions 7.0 and 8.0) issues a fatal error for structures larger than 64K (a holdover from their 16-bit DOS compiler, soon to be fixed). Both translators can convert Fortran COMMON blocks to simple C externals (global data), but a preferable conversion would use structures and pointers. Structures can handle problems with naming conflicts (COMMON variables) and differing COMMON block sizes. The inability to use large structures made it difficult to test my buggy code with Watcom's debugger. I'll cover the Watcom debugger in a future column.

386 SYMBOLIC DEBUGGERS

SVS and Lahey have always offered symbolic debuggers for their 386 compilers. Although powerful, these products are based on the older-style command interface. The modern world has gone to windowing user interfaces, and these debuggers just don't measure up. So I'll examine two new windowing symbolic debuggers: MetaWare now sells their **MDB** symbolic debugger for use with High C-386; MicroWay sells **ClearView-386** for use with NDP C, NDP C++, NDP FORTRAN, and NDP Pascal compilers for the 386/486.

Both companies purchased their 386 windowing debugger code from the same firm, so the two debuggers have a similar user interface. Both use symbolic information inserted into the object and executable files by the compiler and linker. MetaWare's High C requires the Phar Lap 386|DOS-Extender and linker; MicroWay bundles its own linker (based on the Eclipse DOS extender).

MetaWare's MDB requires a recent version of the Phar Lap linker that passes through CodeView debugging information generated by MetaWare compilers (386|LINK-SDB v2.2D or v3.0 of the Phar Lap tools). If you already own Phar Lap's v2.2D, the SDB linker is available free.

MicroWay's ClearView-386 debugger runs under Eclipse's DOS extender and must use the NDPLink utility. The NDP compilers generate COFF-like information for debugging.

(Continued on page 81)

VENDOR INFORMATION

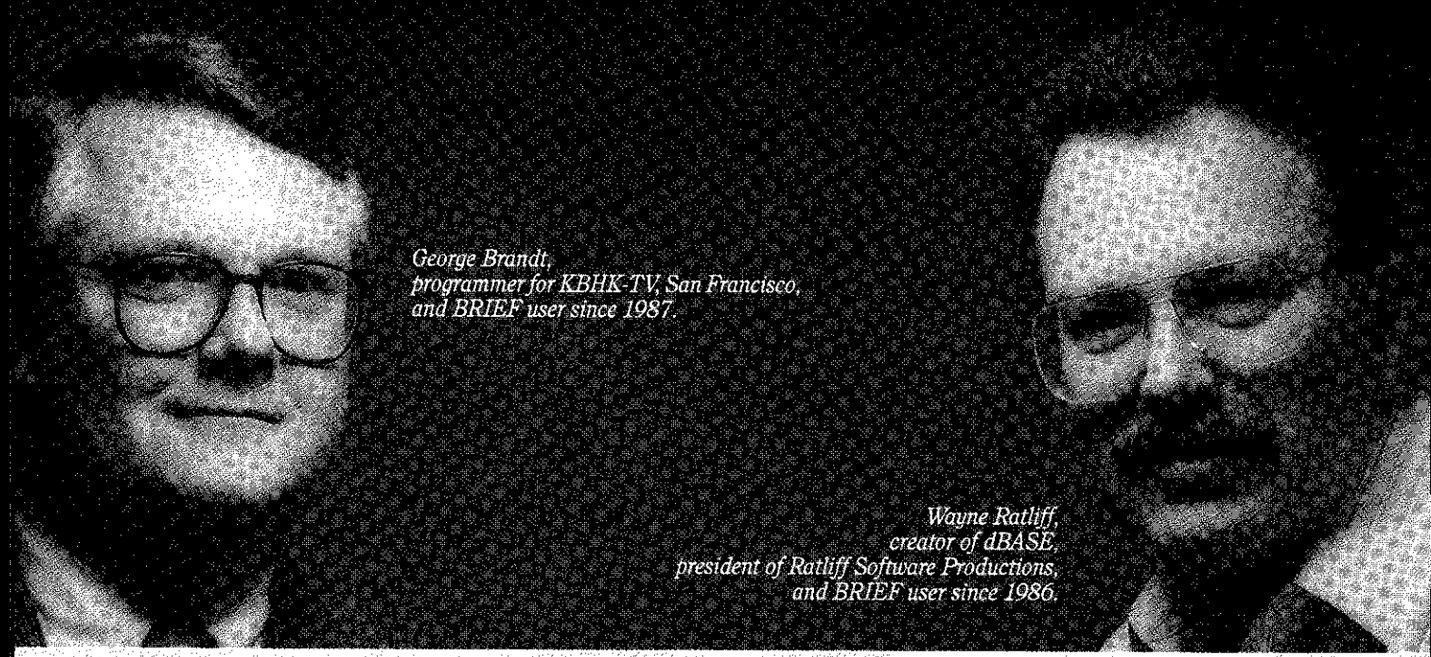
FOR_C v3.1
Cobalt Blue
2940 Union Avenue, Suite C
San Jose, CA 95124
(408) 723-0474
Circle **220** on Reader Service Card

MDB v1.11
MetaWare Inc.
2161 Delaware Avenue
Santa Cruz, CA 95060-5706
(503) 244-8375
Circle **221** on Reader Service Card

ClearView-386 v1.0
MicroWay, Inc.
P. O. Box 79
Kingston, MA 02364
(508) 746-7341
Circle **222** on Reader Service Card

Promula.FORTRAN v3.0
OTG Systems
P. O. Box 239
Clifford, PA 18413-0239
(717) 222-9100
Circle **223** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



George Brandt,
programmer for KBHK-TV, San Francisco,
and BRIEF user since 1987.

Wayne Ratliff,
creator of dBASE,
president of Ratliff Software Productions,
and BRIEF user since 1986.

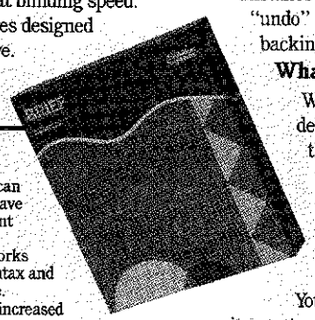
Of course they need completely different kinds of editors. That's why they both use BRIEF.

Let's face it. There's no such thing as two programmers who think alike. Or who work alike.

Because everybody's got their own style, their own habits. Their own quirky little requirements. As well they should. So, where are you going to find one editor that satisfies everyone? Simple. Look inside a BRIEF® box.

It's the right editor, right out of the box.

What you'll find inside is an enormously powerful, astonishingly easy, and highly-evolved program editor. One that works at blinding speed. And that's loaded with features designed to help you be more productive. One guaranteed to give you excellent results within one short hour of booting up the first time.



You'll find features like automatic indenting, infinite windowing, and powerful regular expression searching that goes well beyond the ordinary text-matching you're probably used to.

You'll also find the features that made BRIEF famous. Including our near-miraculous UNDO, that doesn't just un-delete your last keystroke. It actually reverses the effect of any command. So you can explore, experiment, even make gigantic mistakes with hideous consequences, and just "undo" them. And then continue to undo, backing-up through 300 steps.

What you want is what you get.

What you'll also find is an editor that delights in adapting itself to you. With the simple, menu-driven SETUP program, you can quickly customize indenting style, support for any of 36 compilers, keyboard configuration, rate of keyboard auto-repeat, and much more.

You can even edit BRIEF's macros to suit your taste. Or write your own in a snap.

Computer Language said that BRIEF is capable of "almost complete customization." And with new release 3.0, BRIEF is even more powerful and flexible.

For every programmer, it's a different BRIEF.

At last count, there were over 52,000 copies of BRIEF in the field, making life easier and working time more productive for all kinds of programmers—working in all kinds of languages. Programmers as different as Wayne Ratliff is from George Brandt. Which means there are over 52,000 different ways of working, and BRIEF fits them all.

Why not get BRIEF going to work for you? There's no risk. In fact, if you're not completely satisfied that BRIEF is exactly the editor you've been looking for, just return it within 30 days, and we'll return your money. No questions asked.

New BRIEF 3.0 highlights

- With our new C-like macro language, CBRIEF, C programmers can write macros on the fly and not have to switch between C and a different macro language.
- The new source level debugger works with both the original BRIEF syntax and the new CBRIEF macro language.
- Maximum lines per file has been increased to 4 billion.
- We've added to and improved template editing and smart indenting for ADA, Cobol, BASIC, FORTRAN, Modula-2, Pascal, and C.
- We've added editable, multiple keystroke macros that can be saved and restored.
- You can even save and restore your window set-ups.

Plus the features that made BRIEF famous.

- 300-level UNDO safety net
- Flexible windowing, unlimited quantity
- Unlimited number of files
- File size limited only by disk space
- Regular expression search and replace
- BRIEF's LISP-like macro language
- Toll-free tech support

BRIEF 3.0

The Programmer's Editor

Solution Systems™
541 Main St., Suite 410
South Weymouth, MA 02190
FAX 617-337-7719

Try BRIEF at our risk. $249 plus $6 shipping (U.S. and Canada). 30-day money-back guarantee.

Name _____
Company _____
Address _____
City _____ State _____ Zip _____
Phone _____

Paying by ☐ Check ☐ MC ☐ VISA
Card # _____ Exp. date _____
Signature _____
Disk size ☐ 5-1/4" ☐ 3-1/2" Format ☐ DOS ☐ OS/2

Mail this coupon or call toll-free today.

800-821-2492
Extension 496
Massachusetts & outside U.S. call 617-337-6963

Circle **53** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Hardware help

By M. Steven Baker

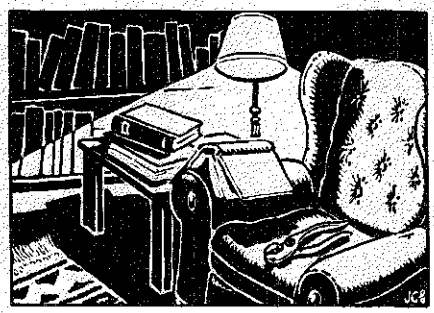


ILLUSTRATION BY JIM CARPENTER

Staying abreast of computer hardware remains a considerable challenge for most programmers. Although PC hardware is not innately complex, the documentation for many common components (for example, disk controllers and network cards) is either poor or difficult to find. Books on various hardware-related issues can help to fill in the gaps left by obscure or obtuse documentation. A tip of the hat to Hal Hardenbergh, who contributed his assessment of two titles (featured under "Computer Architecture") to this group of reviews.

A HELPING OF PC HARDWARE

In its wisdom, IBM chose to publish technical reference manuals on both PC hardware and BIOS software in the early days of PC development. Unfortunately, IBM forgot to document the Intel 8259 interrupt controller. Because I'd used 8259s on old CompuPro CP/M-80 S-100 machines, this oversight was never a problem for me. But for most folks, a decent book on PC hardware is a worthwhile acquisition.

An older book that I recommend heartily is the revised 1986 edition of ***The IBM PC from Inside and Out*** (Addison-Wesley), by Murray Sargent and Richard Shoemaker. A good in-depth reference on standard PC hardware, the book starts with introductory material on PC assembly language and basic digital circuitry. Other chapters focus on interrupts, keyboard and basic video display, data communications, and controlling and monitoring devices. One of the most useful chapters covers the PC's inner workings, including the 8255, the timer/counter, the DMA controller, and the neglected Intel 8259. Because of the book's age, only monochrome, Hercules, and CGA video cards are detailed. In spite of this limitation, few publications discuss in comparable detail the low-level aspects of interrupt controllers, video, keyboard signals, and interfacing.

An area that has never been fully documented by IBM is the AT's bus design. The AT bus is now commonly called Industry Standard Architecture (ISA) to differentiate it from IBM's Microchannel Architecture (MCA). Fortunately, both the AT and the XT have been so widely cloned that de facto standards have arisen. The IEEE P996 Working Group has also developed a standard for the AT bus in the absence of an IBM standard.

Annabooks is a small PC technical publisher located in San Diego ([800] 462-1042). They have two hardware-related titles that I recommend. Published in 1990, their ***AT Bus Design,*** by Edward Solari, is an excellent reference for the AT bus and the ISA portion of the EISA bus. Solari has worked for both Intel and HP and has been an active member of the P996 Working Group. This book provides hardware designers with detailed technical information on the AT bus. The introductory chapters provide an architectural overview and discuss memory and I/O address space, generic bus cycles, and the interaction of ISA and EISA add-on cards. Other chapters detail physical (mechanical) characteristics, signal definition, and the bus cycles—access, transfer, refresh, and arbitration. Solari also discusses performance issues, configuration space, power, and initialization. The book's material is well organized, and many useful tables and timing diagrams are also included.

The second Annabooks title is ***The XT-AT Handbook,*** by Choisser and Foster (1990). This shirt-pocket reference for PC hardware and software contains the kind of information that often sends programmers rummaging through books and manuals. The hardware facts covered include diagnostic error numbers and beep codes, cable and pin connector information, a bus signal summary, card dimensions, switch settings, memory and I/O port maps, hardware interrupts, DMA channels, interrupt maps, AT CMOS memory, AT hard-drive types, video information, and programming data on an assortment of PC chips. Useful software information includes BIOS entry points and data areas, keyboard scan codes (XT, AT-84, and AT-101/102 keyboards), and a DOS command summary that offers help on some of the DOS commands. *The XT-AT Handbook* is an exceptionally useful pocket reference.

Addison-Wesley's ***The IBM PS/2 from the Inside Out,*** by Tom Shanley (1990), provides a hardware introduction to the PS/2 and MCA. Shanley's conversational style enriches the book's tutorial aim. Complex concepts are explained with the help of creative analogies (for example, dishwasher cycles are used to describe bus states). The most valuable of the book's nine major sections is the one on the MCA. Other sections cover different hardware components and issues such as bus cycles, DMA, memory, and peripheral devices. Weaknesses include the use of unusual terminology and the lack of software examples. The term "interrupt slot number" is used instead of the more familiar "interrupt vector," and there is no assembly language listing to illustrate how a hardware register is used. However, if you're just beginning an exploration of PS/2 hardware, this book could be a good starting point.

COMPUTER ARCHITECTURE

The second edition of ***The Art of Electronics,*** by Paul Horowitz and Winfield Hill (Cambridge University Press, 1989), is an excellent reference. A cross between an encyclopedia and a how-to manual, *The Art of Electronics* has been Silicon Valley's best-selling title on hardware electronics. It covers both analog and digital electronics well.

Horowitz and Hill point out that if

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

you ground /DTACK on the 68000 microprocessor, it runs as fast as possible; then they mention a publication called ***DTACK Grounded***. Hmmm. If you have a serious interest in electronics hardware, you should consider this book for your reference library.

Computer Architecture: A Quantitative Approach, by John Hennessy and David Patterson, was published in 1990 by Morgan-Kaufman Publishers. This is the best (and best-selling) book written on computer architecture. David Patterson originated the term "RISC" at U.C. Berkeley and is the godfather of the SPARC. John Hennessy is chief scientist at MIPS and teaches at Stanford. Designed as a college text, this book has been strengthened by having been "beta-tested" in their courses (and at other colleges).

This is the first book on computer architecture to emphasize a quantitative rather than a qualitative approach. The outstanding chapter on caches is the best single source on that subject. Each chapter ends with sections on "Fallacies and Pitfalls" and "Historical Perspective and References."

The book is not perfect. Although it's new enough to contain both wafer and chip photographs of the 80486, the second chapter ("Performance and Cost") uses the 486 as an example of a chip that's too large to be cost-effective. The next-generation SPARC and MIPS chips will be larger than the 486, so I expect this chapter to be drastically revised in a second edition. Its imperfections aside, I strongly recommend this book for those who have an interest in computer architecture.

—Hal Hardenbergh

DEBUGGING WIZARDRY

Published last year by Hayden Books, Tom Swan's ***Mastering Turbo Debugger*** is my pick for the best book on debugging. The clear, concise text provides in-depth coverage of Borland's Turbo Debugger. The first section is a 200-page guide to the debugger and its various utility programs, explaining in detail every command and feature. The sections on the "Art of Debugging" and "Advanced Topics," which comprise the latter half of the book, are exceptional. Along the way, Tom identifies common bugs and describes how to implement successful debugging strategies. Separate chapters using Pascal, C, and assembly language examples are provided for illustration.

Advanced debugging topics are well covered, including remote and dual-monitor debugging, hardware-assisted debugging, and debugging TSRs, interrupt routines, and device drivers. The sections on using the 386 debugger (TD386) and hardware-assisted debugging are most useful. The final section covers data structures—again illustrated using C, C++, Pascal, and assembly language. This hefty book's only flaw is its failure to cover the Turbo Profiler. If only there were equivalent versions of *Mastering Turbo Debugger* for CodeView and the new 386 symbolic debuggers. PJ

(Continued from page 78)

Because these two debuggers are so similar, I'll describe features they have in common and note any differences. Both products use a combination of pull-down menus, function keys, and similar text-command options. Several windows can be displayed on the screen; these windows display registers, variables, debug status, source code, dialog output (results from commands), and a command window. Both single and dual monitors are supported. Remote debugging is available only with MDB. EGA 43-line and VGA 50-line modes are available. An environment file can be used to set up the wide variety of configuration options. Mouse support is partially implemented in MDB and is disabled in ClearView.

Pull-down menus are activated via the familiar menu bar. Text can be scrolled within a window, and window size can be controlled with function keys (MDB has limited mouse support). Windows can be tacked on to the screen with MDB. The debuggers also support user-defined macro keys. Additional features are journal files (for recording and playback of keystrokes) and the ability to list and dump files from the screen.

MDB provides 32 software breakpoints while ClearView limits breakpoints to 5. Both products can also use the four 386 debug registers (which Microsoft's CodeView doesn't yet use) to allow programs to run at full speed until a particular condition occurs. This feature speeds up problem location. Both debuggers support the 387 registers, but only ClearView supports the Weitek registers. Both MetaWare and MicroWay also bundle an execution profiler with their debuggers. MetaWare's MDB v1.11 costs an additional $100 when purchased with the HIGH C compiler, and $295 without it. MicroWay's ClearView-386 v1.0 is $395.

Of the two debuggers, MDB is much more polished. With MDB, more dialog commands have been added to pull-down menus, a mouse is supported, and windowing is more flexible. Both debuggers start to deliver to the 386 DOS environment what programmers expect in DOS tools. By the way, although the best C execution times were fast, they did not quite measure up to the fastest Fortran times. PJ

Imagine...

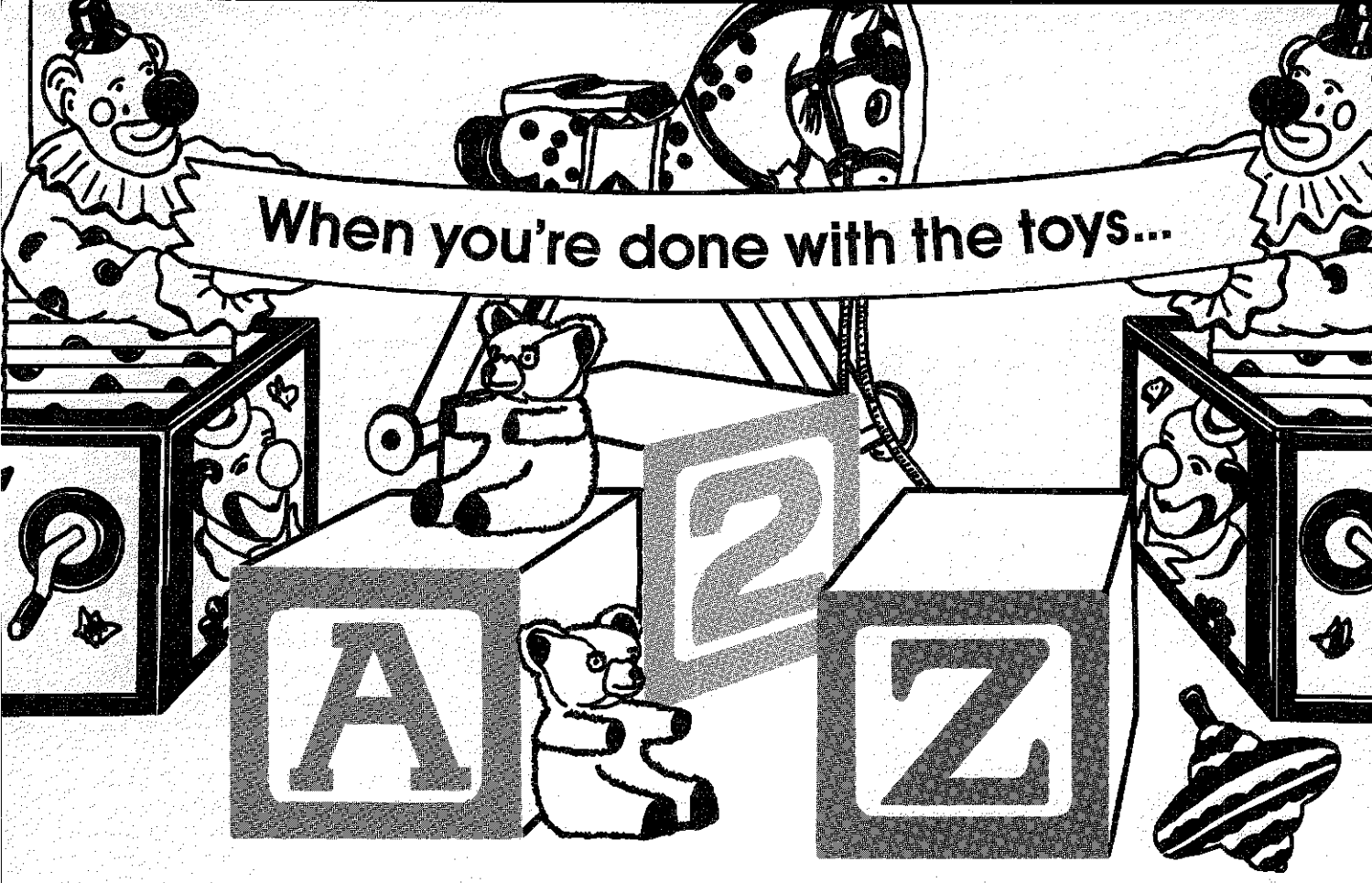
Information on Demand

Imagine having the information you want—at your fingertips. Now, imagine that it's free.

Valuable information on products and services can be yours for FREE—we even pay the postage! Just tear out the ***PJ*** Reader Service Card opposite the Ad Index on p. 72. Circle the numbers from the ads or editorial mentions of the items you want to know more about.

Imagine yourself filling out the card and sending it in today.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



Janus/Ada Professional Development Systems are ready.

Janus/Ada 80386/80486 Unix

POSIX compliant
ACVC 1.11 Validated
Full 32-bit compiler
Extensive Runtime Libraries
80386 Assembler and Disassembler
Janus/Ada Debugger
Interface to C
JWindows
Syntax Checker
Program Profiler

ONLY $900.00

Janus/Ada 80386 MS-DOS

ACVC 1.11 Validated
Hosted in 80386 protected mode
Targeted to the standard DOS environment
Extensive Runtime Libraries
80X86 Assembler and Disassembler
Janus/Ada Debugger including Periscope
Janus/Ada Workstation Environment
Interface to MASM
JWindows
Syntax Checker
Program Profiler

ONLY $500.00

RR Software Inc.

Circle 54 on Reader Service Card

VALIDATED Ada

THIS PRODUCT CONFORMS TO ANSI/MIL-STD-1815A AS DETERMINED BY THE AJPO UNDER ITS CURRENT TESTING PROCEDURES

P.O. Box 1512
Madison, Wisconsin 53701
U.S.A.

Telephone: 608-276-6022
Facsimile: 608-276-6009
Order line: 800-722-3248

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PRESSROOM

MODEL IV DEBUGGER BOARD

Periscope's new Model IV (Revision 2) debugger board runs in 80386 and 80486 systems with CPU speeds of up to 33 MHz. The Model IV pod (CPU interface board) runs in 80486 systems. The hardware comes with Version 5.1 of the Periscope software, and with either a 4K or 16K CPU-event hardware trace buffer. Model IV allows real-time debugging. The new board is $2,495 (4K RAM) or $2,995 (16K RAM); the 80486 pods are $700 each.

The Periscope Company, 1197 Peachtree St., Plaza Level, Atlanta, GA 30361, (800) 722-7006.

Circle **230** on Reader Service Card

LOCATE AND TDREM PACKAGES

Paradigm Systems has released new versions of its LOCATE and its Turbo Debugger Remote Interface (TDREM). These products supply interfaces required to use in-circuit emulators and EPROM programmers with Borland's Turbo Debugger. LOCATE 3.0 ($395) accepts relocatable output from Microsoft C, Turbo C, or Turbo C++ and converts it to an absolute format with optional debugging information. TDREM 2.0 ($195) emulates the Turbo Debugger remote interface, allowing users to observe the operation of Turbo C++ and Microsoft C applications on target system hardware.

Paradigm Systems, 3301 Country Club Road, Ste. 2214, Endwell, NY 13760, (800) 537-5043.

Circle **231** on Reader Service Card

CHARGE DEVELOPMENT TOOL

Solution Systems now has CHARGE, the Software Performance Analyst. CHARGE is a profiler that can give an overview of a system's performance by analyzing how much runtime each routine uses. The information, in the form of bar graphs, statistics, execution counts, and time measures, can be printed or saved to disk.

CHARGE supports Microsoft C 5.1 and later versions, Microsoft QuickBASIC, Borland Turbo C and C++, Turbo Pascal, MASM, and TASM. It interfaces directly with BRIEF, the editor from Solution Systems. CHARGE is $99.

Solution Systems, 372 Washington Street, Wellesley, MA 02181, (800) 821-2492.

Circle **232** on Reader Service Card

NEW DOS-EXTENDER

The new 16-bit 286|DOS-Extender from **Phar Lap Software** provides extended memory for Microsoft C DOS applications. It allows use of the Microsoft compiler, CodeView debugger, and Microsoft linker in protected-mode applications. The 286|DOS-Extender enables users to run character-based OS/2 applications under MS-DOS and supports the XMS, VCPI, and DPMI standards as well as DLLs. It functions as a layer between DOS and your application program; DOS and BIOS calls are made via software interrupts. The 286|DOS Extender SDK Version 1.2 is $495. Phar Lap's Version 3.0 of the 386|DOS-Extender SDK for 32-bit applications is $495.

Phar Lap Software, 60 Aberdeen Avenue, Cambridge, MA 02138, (617) 661-1510.

Circle **233** on Reader Service Card

TURBO PASCAL FOR WINDOWS

Borland International's Turbo Pascal for Windows provides a self-contained environment for developing programs to run under Windows 3. Included are a DOS command-line compiler, six volumes of documentation, and Object Windows. Object Windows, which is similar to the company's Turbo Vision, eases the development process of Windows programming. WinCRT and WinDOS, which replace the CRT and DOS units, help in porting existing DOS applications to Windows. The package can use DLLs from Borland C++, Paradox Engine 2.0, and others. Turbo Pascal for Windows costs $249.95; upgrades are $99.95.

Borland International, 1800 Green Hills Road, Scotts Valley, CA 95067-0001, (800) 331-0877.

Circle **234** on Reader Service Card

TECHNOJOCK OBJECT TOOLKIT

TechnoJock has released TechnoJock Object Toolkit for Turbo Pascal, a collection of objects designed for use by those who are new to object-oriented programming. The Toolkit provides help with screen and window management, forms input, list management, menus, and directory and file selection. TechnoJock Object Toolkit requires Borland Turbo Pascal version 5.5 or later. The Toolkit is $89.95, with a special price of $69.95 through May 30. Upgrades are available for registered users.

TechnoJock Software, P.O. Box 820927, Houston, TX 77282-0927, (800) 242-4775.

Circle **235** on Reader Service Card

JPI TOPSPEED COMPILERS

Jensen & Partners International (JPI) has announced the addition of C++ and Pascal to their TopSpeed Multi-language Development System. The system allows users to enter, edit, compile, debug, and run programs using any combination of TopSpeed's four programming languages. Available compilers include version 3.0 of TopSpeed C and TopSpeed Modula-2, as well as the new C++ and Pascal units. All of the compilers are available as separate modules. They support programs that run under DOS, Windows 3, and OS/2, and all have a multilanguage, multiplatform base. The TopSpeed system modules are priced at $99 each.

JPI has also inaugurated the **TopSpeed Consortium,** a network of third-party software vendors that support the TopSpeed product line. The consortium offers sales and marketing support to members.

Jensen & Partners International, 1101 San Antonio Road, Ste. 301, Mountain View, CA 94043, (415) 967-3200.

Circle **236** on Reader Service Card

WATCOM C/P16 COMPILER

Watcom's C/p16 Compacting and Optimizing Compiler is designed to reduce code size for a variety of application programs. The compiler executes compacted 16-bit pseudo object (p16) code, resulting in overall application size reductions as great as 40%. Features include the WVIDEO debugger, a linker, an execution profiler, a MAKE, and a librarian. The Watcom C/p16 compiler, available in June, is $5,000, and includes a royalty-free license for runtime support.

Watcom, 415 Phillip Street, Waterloo, ON, Canada N2L 3X2, (800) 265-4555.

Circle **237** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PROTOVIEW AND PROTOGEN

ProtoView has introduced version 3.10 of the ProtoView Screen Management Facility ($695). The package provides data entry field control objects, field control macros, dynamic data exchange support, and high-level printing and animation functions. There are no proprietary languages used, runtime royalties, or fees for ProtoView applications. Also new is the ProtoGen Application Generator ($199), which allows users to specify a top-level window that contains controls, icons, and bitmap objects that can launch other screens and programs. ProtoView and ProtoGen generate code that is C++ compatible.

ProtoView Development Co., 353 Georges Road, Dayton, NJ 08810, (908) 329-8588.

Circle **238** on Reader Service Card

VZ PROGRAMMER

VZ Corp. has announced VZ Programmer version 2.0, an object-oriented programming platform that supports either the OS/2 Presentation Manager or Windows 3. Version 2.0 supports C and C++ and incorporates a GUI drawing tool, an OOP database, and a debugger. VZ Corp products use the company's Visu*lize technology, enabling programmers and end users to develop interactive programs that use graphical presentation. VZ Programmer for Windows 3 costs $595, and $1,295 for OS/2.

VZ Corp., 57 West South Temple Street, Salt Lake City, UT 84101, (801) 595-1352.

Circle **239** on Reader Service Card

ZINC SUPPORTS WINDOWS

Zinc Software has introduced version 2.0 of the Zinc Interface Library. Version 2.0 allows programmers to create applications for MS Windows, as well as for DOS Text and DOS Graphics modes, from a single set of source code. Both C++ and object-oriented technology are supported. Version 2.0 includes an interactive design tool to streamline GUI. The DOS version of Zinc Interface Library costs $199 ($249 with source). The combined DOS and Windows version is $299 ($399 with source).

Zinc Software Incorporated, 405 South 100 East, Ste. 201, Pleasant Grove, UT 84062, (800) 638-8665.

Circle **240** on Reader Service Card

SMALLTALK/V FOR WINDOWS

Digitalk's new Smalltalk/V Windows ($499.95) provides DOS programmers access to a next-generation prototyping and development environment. Smalltalk/V combines the company's OOP environment with Microsoft Windows 3. Features include browsers, inspectors, debuggers, and DLLs, as well as interfaces to DDE. Smalltalk/V Windows source code is compatible with the company's Smalltalk/V PM programming environment.

Digitalk, Inc., 9841 Airport Blvd., Los Angeles, CA 90045, (213) 645-1082.

Circle **241** on Reader Service Card

DIALOGCODER GENERATOR

The Software Organization, Inc., has introduced DialogCoder ($499), a CASE tool and C source code generator for the design, creation, testing, and modification of Windows 3 user-interface dialog boxes. DialogCoder features a dialog template file, a point-and-click interface, and an animation feature for dialog testing. The program generates code to support everything from flat WYSIWYG dialogs to more complex types of dialogs.

The Software Organization, Inc., 56 Kirkland Street, Ste. 3, Cambridge, MA 02138, (617) 354-2012.

Circle **242** on Reader Service Card

THE CREATOR VERSION 6.0

TNT Software's The Creator 6.0 is a QuickBASIC database and report form generator, with a documentation generator and data dictionary utility. All native QuickBASIC data types are supported. Records can be up to 20K in length and have 11 key fields; a file can contain as many as 2 billion records.

The Creator version 6.0 ($99) adds a modified version of TNT Software's My Word ($49) for text edits, network capability, invisible fields, and batch lookup and update. Eighty QuickBASIC library routines are supplied, and programs can call routines written in other languages. No royalties are required for generated code or runtime routines.

TNT Software, Inc., 34069 Hainsville Road, Round Lake, IL 60073, (708) 223-8595.

Circle **243** on Reader Service Card

NEW PROGRAMMER'S TOOLBOX

MMC AD Systems has released Volume III of its C Programmer's Toolbox for MS-DOS. Volume III's 13 new tools span the areas of translation, file compression and archiving, memory use, and miscellaneous user interface programs. TSYM, a TSR tool, adds symbolic links to DOS that are similar to those in Berkeley Unix. One of the translation tools, CDecl, translates C declaration statements and cast operators from C to English or English to C. The Toolbox works with most C dialects and with any size program. It also complements any existing programming environment. Volume III is $125.

MMC AD Systems, Box 360845, Milpitas, CA 95036, (415) 770-0858.

Circle **244** on Reader Service Card

VERIFY WITH ASM CHECKER

V Communications has released ASM Checker, an assembly source code verification tool that looks for possible bugs and coding errors. ASM Checker provides a complete cross-reference of external and public symbols, identifying symbol types and mismatches of types across multiple files. ASM Checker costs $179.95.

V Communications, 4320 Stevens Creek Blvd., Suite 275, San Jose, CA 95128, (800) 648-8266.

Circle **245** on Reader Service Card

DIFF.IT 2.0 COMPARISON TOOL

Double R Software has now released Diff*it 2.0, a file comparison tool. Diff*it compares two files line-by-line, word-by-word, or byte-by-byte (hex mode). New features include full mouse support, a mixed-file mode disk output, and enhanced resync search algorithms. Screen output can be minimal or full. The menuing interface comes with pop-up help screens. Diff*it costs $129.95.

Double R Software, RD 1 Fifth Avenue, Valley Forge Manor, Phoenixville, PA 19460, (215) 666-6104.

Circle **246** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ BILLBOARD

FIGHT PIRACY & PROTECT YOUR PROGRAM $$$'s!

Since 1986, companies worldwide have been choosing Az-Tech security products. If you demand the strongest protection available, why not choose one of these "proven leaders":

- **EVERLOCK** Copy Protection
- **EVERTRAK** Software Security
- **EVERKEY** Hardware "Key" Software Security

For IBM and Compatibles.
30 day money back guarantee.
Free info and demo disk available.

Az-Tech Software, Inc.
305 East Franklin
Richmond, MO 64085
(800) 227-0644 (816) 776-2700
FAX (816) 776-8398

Circle 110 on Reader Service Card

QuickBASIC 4.5 - PDS 7
Software Development System

- Multiple Overlapping Windows with Full Mouse Support.
- Windows can be clicked, moved, resized, scrolled, etc.
- Popup & Pulldown Menus, Lists, Calculator, Calendar, Directories, and more.
- Dialog Boxes and Data Entry Screens.
- Context Sensitive Help Screens.
- Event-Driven Architecture.
- Supports MDA-VGA text modes.
- Integrated Development Environment.
- File, Directory, Keyboard, BIOS, and TSR functions.
- Disk-based Documentation.
- Only $59.95 (+CA Tax & $5.00 S/H)

DSE Software Publishing
Post Office Box 96 - Willits, CA 95490
707 459-4358

Circle 137 on Reader Service Card

Duplicating disks? Read this:
If not, please forward to someone who is.

EZ-DISKCOPY PRO
Duplicates Disks Fast!

- Read source disk only once.
- Quickly & flawlessly duplicates 360K, 720K, 1.2M, 1.44M disks on your own PC/XT/AT/etc. ©
- Up to 300 disks/hr/machine!
- Formats, copies, verifies, stores images (optionally "crushed"), etc.
- Pre-formatted are even faster.
- More Speed? Just add machines.
- Serialize and print labels too!
- Replaces $1000s in hardware.

$139+S&H (CP.1; $69ea. add) $495 NCP.10

EZX, P.O. Box 58177, Webster, TX 77598

Discover, VISA, MC, AX Accepted
713.280.9900 FAX:280.0525
Call for Info FAX-BACK (USA/Cda) or a TestDrive

Circle 111 on Reader Service Card

Write Multitasking DOS Programs with Conquerrent C Function Library

- Execute up to 16 C functions concurrently within one program.
- Great for real-time, data acquisition, clock displays, background printing, simulation, process control, games.
- Transparent reentrancy protection for most library functions.
- All memory models supported in Turbo and Microsoft C on DOS 80x86 PC's.
- 4K to 12K overhead.
- $99 complete, no royalties, 30 day satisfaction guarantee.

Available from:
Interstitial Software Systems
321 E. Buckingham Rd., Ste. 140
Garland, TX 75040

Circle 112 on Reader Service Card

'Quick and Clean' COBOL®

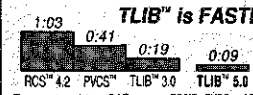
The COBOL Report Activator *(COBRA)*:

- ☆ Generates efficient, structured COBOL programs
- ☆ Is easy to learn and use
- ☆ Accepts your COBOL code and copybook members
- ☆ Produces audit trail reports
- ☆ Uses one or two file input
- ☆ Allows site defined defaults
- ☆ Available in mainframe & PC

Olivadoti Associates
(Formerly *COBRA* Systems)
P.O. Box 750817
Petaluma, CA 94975-0817
(707)763-5123

Contact this advertiser directly

TLIB™ is FASTEST!



Times are to update a 45K library on a PC/XT. PVCS and TLIB 3.0 are from Sept 87 *PC Tech Journal*. MKS RCS 4.2 and TLIB 5.0 are newer.

TLIB™ is *BEST!*

"Do not be fooled by the fact that this is the least expensive of the five packages reviewed here - TLIB has features and power to spare" John Rex, **Computer Language**

"TLIB is a great system" J. Vallino, **PC Tech J**

- **Full-Featured Version Control for Software Professionals.** Check-in/out locking. Branching. Keywords. Wildcard and list-of-file support. Can merge parallel changes and undo intermediate revisions. Network and WORM support. **Mainframe compatible** deltas for Pansophic, ADR, IBM, etc.. Integrates with **Opus™ MAKE** & **Slick™ MAKE**.

MS-DOS $139, OS/2 $195 + shipping Visa/MC
5 station LAN license $419 (OS/2 $595), call for other sizes

BURTON SYSTEMS SOFTWARE
PO Box 4156, Cary, NC 27519 **(919) 233-8128**

Circle 113 on Reader Service Card

Why you want BATCOM!

BATCOM is a batch file compiler that compiles your ".bat" files to ".exe" files to make them faster, more professional, and more capable. **BATCOM** extends DOS with new commands so you can read keyboard input, perform arithmetic, use subroutines, and much more. In addition, **BATCOM** protects your source code, and you can distribute your compiled programs without royalties. Only **$59.95**

ORDER TODAY!

Wenham Software Co.
5 Burley St.
Wenham, Ma. 01984
(508) 774-7036

Circle 114 on Reader Service Card

FREE TRIAL

Pinnacle File Manager

We guarantee immediate productivity gains from using our C file manager library. Simple table-and-column view of data with advanced B-Tree indexing and hi-speed cache frees your brain to solve application-specific problems.

Vermont Database Corp.
RR3 Box 6516
Stowe, VT 05672
802-253-4437

Contact this advertiser directly

FORTRAN

FORTRAN Toolkits
Comprehensive libraries of over 300 graphics, text, and systems service subroutines from ellipses to child-processes to I/O verification. High-speed video drivers, from CGA to Super VGA, are all internal. Option rich, easy to use. No royalties. 300+ pg. manual, 8,000 lines of sample source. **Price $215.**

DOCUMENT'er and DIAGRAM'er
Makes source code understandable. Indexes, cross-references, and diagrams the structure of source code. No changes to source required. Nested includes, recursion, and annotations handled automatically. **Price $165.**

IMPULSE *Engineering*
PO Box 190206,
San Francisco, CA 94119-0206
(415) 788-4611

Circle 115 on Reader Service Card

THE CD-ROM DEVELOPER'S LAB™

Comprehensive CD-ROM multimedia production resource for PC and Apple developers. Proven design, management, data prep, programming, premastering and manufacturing techniques. Important specs from leading companies. Demos of off-the-shelf tools for imaging, audio and animation. Sample applications demonstrate the power of our source tools in C, Turbo Pascal.
PC or MAC.@ **$395.00;**
Transportable version @ **$425.00,**
Visa/MC accepted.

SOFTWARE MART, INC.
3933 Spicewood Springs Rd., Suite E-100
Austin, Texas 78759
(512) 346-7887 FAX: (512) 346-1393

Circle 116 on Reader Service Card

"Model the World" With Class

MEIJIN++
C++ Modeling & Simulation Class Library

- Reusable Data Structures
- Statistical & Math Classes
- Configurable Discrete-Event & Scheduling Mechanisms
- Handling of Exceptions & Virtual Memory Paging
- C++ 2.0 Documented Source
- No Royalties. User's Manual
- Introductory Price $225.00

Support for Zortech 2.0+ & Borland TC++ 1.0+

NETWORK INTEGRATED SERVICES, INC.
714-755-0995
221 West Dyer Road
Santa Ana, CA 92707
Fax: (714) 433-2347

Circle 117 on Reader Service Card

INTEL® 80486 MACRO DISASSEMBLER

MD86 (Masterful Disassembler) is the most comprehensive macro disassembler on the market. Interactive by design not after-thought. Features include:

- ▸ User defined macros with arguments
- ▸ Auto code/data separation
- ▸ Auto commenting and label naming
- ▸ Command menus and help screens
- ▸ Customizable disassembly options
- ▸ Comprehensive, indexed, manuals
- ▸ Fully supports 8086/87 thru 80486/87
- ▸ COM/EXE/SYS/memory/other
- ▸ Cross reference and more!

Still only $67.50+tax ($1.50 s/h)

C.C. SOFTWARE
1907 ALVARADO AVE.
WALNUT CREEK, CA 94596
(415)939-8153

Circle 118 on Reader Service Card

Offer your product directly to Professional Programmers:

Placing an ad in **PJ BILLBOARD** is an easy, low-cost way to reach potential buyers. Display ads are 1⅝" wide by 2⅜" high. For more information or to reserve a space, call our ad department at **(800) 234-0386.**

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ BILLBOARD

POWER GRAPHICS PROGRAMMING

If you do EGA/VGA programming, then *Power Graphics Programming* is a "must-have" for your reference library. This Que book is a compilation of Michael Abrash's first four EGA and nine VGA "On Graphics" articles published in *Programmer's Journal*. It contains great theory, invaluable tips, and nuts-and-bolts ASM and C code listings. A companion code disk is available. Order your book and disk today.

P.G.P. Book .....
P.G.P. Code Disk .....
*Plus $1.50 ha... per.
Free shi... U.S.
Add'l ... to Canada,
Mexico ... foreign destinations.

SEE ISSUE SPECIAL PAGE 73

Order your book and disk direct from:
Programmer's Journal
Call toll free with your VISA/MC
800-234-0386
Outside the U.S.? Call 503-747-0800

Contact this advertiser directly

TSR MENUS and TSR 'C' Reference

No more looking through books for the correct syntax!

TSR MENUS is a configurable system of versatile memory-resident menus. Each menu selection can be set up to pop up other menus, scrolling information screens, or file viewing windows.

TSR MENUS has been set up with a complete database of information on MSC 5.1 and Quick C. Once installed, it will pop up over your editor with menu selections grouped by type of command.

TSR 'C' REFERENCE $39.00
TSR MENUS setup $89.00

Omni-Tech Software
Seattle, WA
(206) 874-4512

Circle 120 on Reader Service Card

If You Are Honestly Not Interested In Your App's Performance, Then Stop Here

However, if knowing what is occurring and where the time is being spent in your app is important, then you need ***Performance Tracer.***™

Performance Tracer provides comprehensive, function level detail in a single pass — including a complete call tree history and every single exercised function prioritized by execution time and frequency.

The data you need to make informed performance tuning decisions — which functions should be in assembly, which should be nondescardable, the ramification of switching a call from far to near, etc.

Profiles EXE's, DLL's. No source changes. Easy to use. DOS, Windows, OS/2 compatible. Money-Back Guarantee

Also shipping ***SegMentor***™ — the first computer-assisted solution for mathematically optimizing the segmentation map of Windows applications.

CALL OR FAX FOR FREE DEMO DISK

MicroQuill INC. 4900 25th Ave. NE #206 Seattle, WA 98105
Toll Free 800-441-7822 • Fax: (206) 525-8309

Circle 121 on Reader Service Card

DEBUG WITHOUT SOURCE

Quaid Analyzer debugs by showing interrupts as they occur. You can even watch DOS boot load without hardware changes. The manual tells how to deal with programs that don't want to be traced - we use it on copy protection. You scroll through memory like text in an editor and never type a command. US $200.00

Quaid Software Ltd Dept PJ
45 Charles St East Third Floor
Toronto Ontario Canada M4Y 1S2
(416) 961-8243
FAX: (416) 961-6448

Circle 137 on Reader Service Card

A Software Development Essential!

DOCZ™

Save up to 35% of your development time with DOCZ to automate subroutine and program documentation. Take advantage of the object-oriented technology of DOCZ to share and re-use your software modules. FREE MSDOS demo disk and samples. 30-day money-back guarantee. VISA/MC accepted. $195/Single-User. Start benefiting NOW!

Do It Better With Toolz!
Software Toolz, Inc.
8030 Pooles Mill Drive
Ball Ground, GA 30107-9610
(404) 889-8264
Orders, V-mail: (800) 869-3878

Circle 122 on Reader Service Card

ATTENTION Software Writers!

Is your software innovative, usable and needed? Are your applications practical, thorough and well tested? Is your documentation well organized and understandable? InSync, a division of an established international software distributor, is evaluating new and innovative software to publish and market domestically and internationally. We are seeking new off-the-shelf MS/PC-DOS software applications (no accounting or inventory, please). ORIGINALITY IS A MUST for all submissions.

Please call our Evaluations Department to discuss submission procedures.

InSync
soft ware
1-800-783-7053

Circle 123 on Reader Service Card

PC-lint

C Programmers: Locate bugs, glitches, quirks and inconsistancies in your C code. PC-lint supports multiple modules, K&R, ANSI. Specify OS/2 or MS-DOS. Only $139.

Gimpel Software
(215)584-4261
3207 Hogarth Ln
Collegeville PA 19426
30 Day Money-back Guarantee

Contact this advertiser directly

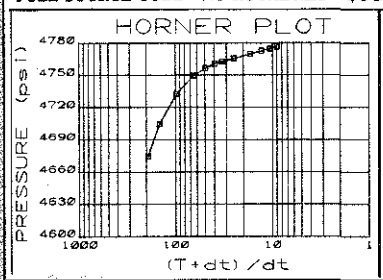
INGRAF 5.00

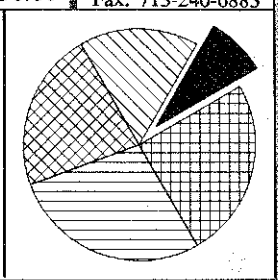
A Multi-Device Graphics Library for Scientific, Engineering and Business applications. Supports Video(CGA, EGA, VGA & Hercules), Printers (EPSON, Okidata, HP LaserJet, HP Paintjet & Postscript) and Pen Plotters(HPGL and DM/PL).

C, FORTRAN-77, PASCAL, MS Professional BASIC and QuickBASIC bindings.

FULL SOURCE CODE • NO ROYALTIES • $350.00

Sutrasoft
10506 Permian Dr.
P.O. Box 1733
Sugar Land
TX 77487-1733
U.S.A.
Tel: 713-491-2088
Fax: 713-240-6883





Circle 125 on Reader Service Card

PJ CODE IS AVAILABLE!

All the source code from the articles in this issue is available on disk from

Programmer's Journal
P.O. Box 70167
Eugene, OR 97401-0110
800-234-0386
503-747-0800

Code disks are $10.00 each. Please see page 73 for ordering information.

PJ Code Diskscriptions are available for $39.95. Look on the card opposite page 16 to order.

The code listings are also available for downloading from several Bulletin Board Systems around the country. See page 72 for listings.

Contact this advertiser directly

Coming Next Issue: NETWORKING & COMMUNICATIONS

July/August 1991 Vol. 9.4
Advertising space reservations are due:
May 3, 1991
Materials are due:
May 10, 1991

Please call 503-747-0800

Ask for Jay in the advertising department to reserve your space.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ BILLBOARD

OPT-TECH SORT/MERGE

Extremely fast Sort / Merge / Select utility. Run as an MS-DOS command or CALL as a subroutine.
Supports most languages and filetypes including Btrieve and dBase. Unlimited filesizes, multiple keys and much more!

MS-DOS **$149.**
OS/2, UNIX **$249.**

Opt-Tech Data Processing
P. O. Box 678
Zephyr Cove, NV 89448
(702) 588-3737

Circle 127 on Reader Service Card

QUALITY TOOLS

Memory Overlay Manager - transform your C, Clipper or Clarion application into a 5K TSR with hot-keys.....$99.95
Memory Master - break 640K barrier, one heap or many, EMS/XMS.....$59.95
ASMplus - structured ASM lang. & lib. Code ASM just like C or Pascal....$99.95
"C"erious Toolkit Plus - over 250 highly optimized fcts, C & ASM source...$99.95
386PM protected mode ver.....$199.95
Phase One - 100% replaceable plug-&-play C engine for Paradox, runs 4-10X faster, DOS and DDL ver. incl.....$495.00
PALCom - PAL lang. compiler. Create stand alone .EXE programs.....$1495.00

Please call our BBS at 516-331-6682 to try out our FREE demo programs.

TSR Systems Limited
116 Oakland Avenue
Port Jefferson, NY 11777
Ph: 516-331-6336 FAX: 516-331-6377

Circle 128 on Reader Service Card

BGI Printer Drivers

The BGI Printer Driver Toolkit provides BGI drivers for Epson 9 & 24 pin, HP LaserJet II, and HP PaintJet printers. Use the BGI graphics interface to effortlessly generate high resolution hardcopy with Turbo C, TC++, and Turbo Pascal. **Full driver source code included.** $89.95

PC Timer Tools

60 functions implement microsecond resolution timing, precision delays, interrupt profiling, and timer tick management. Ideal for profiling and data acquisition/process control. Supports TC, TC++, MSC, TP. **Full library source code included.** $49.95

Prices postpaid USA, elsewhere add $4
VISA & MasterCard accepted.
30 day money back guarantee

RYLE DESIGN
Purveyors of Big Science
PO Box 22, Mt. Pleasant MI 48804
(517) 773-0587 CIS:73047,1765

Circle 129 on Reader Service Card

The $25 Network

Try the 1st truly low cost LAN

- Connect 2 or 3 PCs, XTs, ATs, PS/2s
- Uses serial ports and 5 wire cable
- Runs at 115K baud, up to 90 feet
- Transfer 8500 bytes per second (ATs)
- Runs in background, totally transparent
- Share any device, any file, any time
- Needs only 14K of ram
- Replace all file transfer software
- $25 per network, NOT PER NODE!
- OVER 15,000 SOLD WORLD WIDE

Skeptical? We make believers!

iModes Information Modes
Drawer F, Denton, TX 76202
817-387-3339 Technical
1-800-628-7992 Orders

Circle 130 on Reader Service Card

BASIC AND C PROGRAMMERS

INDEX MANAGER ®
B+ Tree File Indexing
Simple, powerful, affordable

New QuickC/TurboC version $99
QuickBASIC version $59
BASIC PDS 7 version $99
Network version $250

CDP Consultants
1700 Circo Del Cielo Drive
El Cajon, CA. 92020
(619) 440-6482

Contact this advertiser directly

Software Marketing

The *Software Success Reference Book (1989)* is a **MUST READ** if you want to market your software products successfully. Written by David H. Bowen, publisher of *Software Success*,™ the monthly newsletter on successfully running a software business. The *Reference Book* is a 262-page guide organized by topic. Covers Lead Generation, Promotion, Pricing, Distribution, Support, etc. **Only $50.** Check or Credit Card (VISA/MC/AEX).
100% Money Back Guarantee

Software Success
PO Box 9006, San Jose, CA 95157
(408) 446-2504
FAX (408) 255-1098

Circle 131 on Reader Service Card

DISK DISTRIBUTOR

SOFTWARE DISTRIBUTORS
SITE LICENSEES
USERS GROUPS

AUTOMATICALLY DUPLICATE AND LABEL DISKETTES WITH YOUR PC. NO EXTRA HARDWARE REQUIRED.

Use your PC to duplicate software distributions of any quantity of disks AND print labels. Duplicates any combination of 3 1/2" & 5 1/4" disk formats. Stores disk images on hard disk for unlimited copying. NCP.

INTRODUCTORY PRICE $79

ORION SOFTWARE CO.
P.O. Box 1913
Bellingham, WA 98227 U.S.A.
Tel: 206 647-2138 Fax: 206 734-7328

Circle 132 on Reader Service Card

UPPER DECK EDITOR

FOR WINDOWS 3.0

A FAST, flexible, and robust programmer's editor designed for the Windows environment. File size and number of document windows limited only by available memory. $75 plus $5 S/H (outside USA $15). CA residents add sales tax.

UPPER DECK SYSTEMS
P.O. Box 263342
Escondido, CA 92030
(619) 741-1075

Circle 133 on Reader Service Card

The $20 C/C++ Object-Based eXecutive

Try the most comprehensive and cost-effective multitasking executive library available!

OBX can coexist with DOS and allows your PC to execute tasks simultaneously at a simplicity you just wouldn't believe.

OBX is a low cost executive library for software development in an industrial low speed real-time environment.

OBX is also a valuable educational tool for programmers tha want to learn the concepts of real-time programming.

OBX/C	$20
OBX/C w/Source	$90
OBX/C++	$30
OBX/C++w/Source	$140

Cnapse
The Real-Time Contact
11 Laurier, Chambly
Qc, Canada J3L 9Z7
(514) 447-7221
Fax: (514) 447-7297

No royalties, no charge for AIR shipping, and 30-day money back guarantee. All orders must be prepaid in US funds. Send a check or money order payable to Cnapse.

Circle 134 on Reader Service Card

MAKE $$$ FREE

Device Driver Inc. would like to market device driver programs that have been developed by your company. These programs have significant revenue potential.

Drivers are marketed through our catalog, which is distributed at trade shows & through direct mail to systems integrators, manufactures of computer peripherals, VARS, consultants and other interested parties. If you have written drivers for PC's, workstations, midrange or super computers and are interested in having them listed in our catalog, for free, call **(401) 724-8756** or write to:

DEVICE DRIVERS, INC.
990 Mineral Springs Ave.
No. Prov., RI 02904
Fax # (401) 726-8940

Circle 135 on Reader Service Card

Imagine what 16 million new buyers could mean to your software business...

Cash! When you put your new product in the Selective Software Catalog your sales will jump because you get:

- **Buyers:** you get buyers not lookers and you keep the names of your future good customers.
- **Circulation:** we mail 3-4 million catalogs every 2 months.
- **Free ad production:** get pro quality ads at a fraction of the cost of an agency!
- **Service and support:** we handle all sales, top-rate tech. support and fulfillment.
- **Disk dupe & packaging:** we can help you with expert package design & disk duping.
- **Payment systems:** co-op plans for first time developers.

Selective *software*
Call: 408-423-3556 Ext. 200 for more details

Circle 136 on Reader Service Card

ISSUE SPECIAL
Save $10
See Page 73

POWER GRAPHICS PROGRAMMING BOOK AND CODE DISK, $29.90.
For a limited time, you can save over 26% when you order both the *Power Graphics Programming* book and the companion code disk during the ***PJ*** 9.3 Issue Special. Order your book and disk today for only $29.90 and save $10.00.
(Offer expires June 30, 1991. All orders must be prepaid.)

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Hardware decisions

By Hal W. Hardenbergh

ILLUSTRATION BY JIM CARPENTER

My employer's in the image processing business, with a large proportion of that business in what's called "remote sensing." Usually, that means telemetered data gathered by a video camera in low earth orbit. Some of our products are being used to check the wheat fields north of Saudi Arabia for wheat blight. The location and extent of the infestation are currently of great interest.

This past holiday season, I saw an astonishing number of advertisements for home video cameras. Apparently, millions of Americans want a video tape of baby's first step. The technology in those home video cameras isn't a heck of a lot different from that used in remote sensing, but the application of the technology is very, very different.

All of us know instinctively that the consumer marketplace is very different from the industrial or industrial/military market. We usually spend as much time thinking about this difference as we spend thinking about how to breathe.

Now, I'm not claiming to understand the consumer marketplace. If I did, I'd be filthy rich—and I'm hardly a little bit dirty. Fads come and go for no apparent reason. A few years back, teenyboppers wore $75 jeans that were just like the $12 ones except for a signature stitched across the fanny.

We once bought cars with vertical tail fins. Then tail fins died out. Nowadays cars have horizontal tail fins. If you're driving 437 mph, the horizontal tail fins give better rear-wheel traction. At 30 mph on an icy highway, they're about as effective as a butterfly resting on the trunk. Come to think of it, the original tail fins improved directional stability, but folks never seemed to have any trouble making turns.

Obviously, a consumer buying decision (pet rocks) is arbitrary and capricious, whereas industrial buying (machine tools) is far more carefully reasoned. Because this magazine is about PCs, and because PCs are bought for both consumer and industrial purposes, let's look at a specific example in this area.

On this long table we have four AT-style PCs that all use 25-MHz 486s. All have almost exactly the same CPU performance. The one on the left is a no-name clone whose base price without hard disk or display system is $1,489. Add $199 for a 42MB (28-ms) drive, $62 for a second floppy (this one 1.44MB), $150 to expand the memory another 3MB (to total 4MB), $229 for a very good name-brand extended VGA card, and $589 for a NEC Multisync 3-D monitor. Total price, $2,718.

The other three machines are all well-known name brands sold in mainline computer retail outlets. One is made in Fremont, California, and has an $11,898 price tag. Another comes out of Irvine, California, and costs $15,519. The last one is made in Texas and sells for (cough) $28,868. That's more than a ten-to-one price range for very similar computers. The higher-price spreads may have larger hard disks; the $28,000 model has a much larger hard disk and is intended for use as a server. Is the hard disk $26,000 larger? No. Besides, for another $500 you can upgrade to a 200MB (18-ms) hard drive on the no-name clone.

At such a low price, that clone must be cheaply made. Let's see: steel case, Intel 486 CPU, epoxy-glass motherboard with six to eight expansion slots, Taiwanese switching power supply, Teac floppies, brand-name hard disks (NEC, Western Digital, or Seagate), and Japanese DRAMs. The display system is top-quality, brand-name equipment. You know something? This looks a lot like the higher-priced models!

Big corporations like to buy the name-brand models because they like the signature stitched across the fanny, because they believe that the reliability and service of the name-brand are superior, and because the no-name clone dealer expects them to pay cash on delivery.

Reliability? I work in a largish company that has lots of clones and lots of name-brand computers. I don't see any difference in the failure rate, which is very low. If this is truly the motivation, why not buy four or five clones for the same price as one name-brand PC and discard the clones as they fail?

Hmmm. Didn't I just write that industrial buying decisions are made rationally? I seem to be contradicting myself.

CHANGING GEARS

Lately I've had more fun reading the trade journals than the international news. This is the time of year we get the scorecards for the previous year and the infallible expert predictions for the next year. Let's see: Intel jumped from eighth to fifth in international semiconductor sales, thanks in large part to its 386. It appears that Roger Ross's prediction of last summer that RISC sales would increase by 400% in one year is gonna fall about 380% short. And I've lost count of the number of experts predicting the imminent demise of DOS.

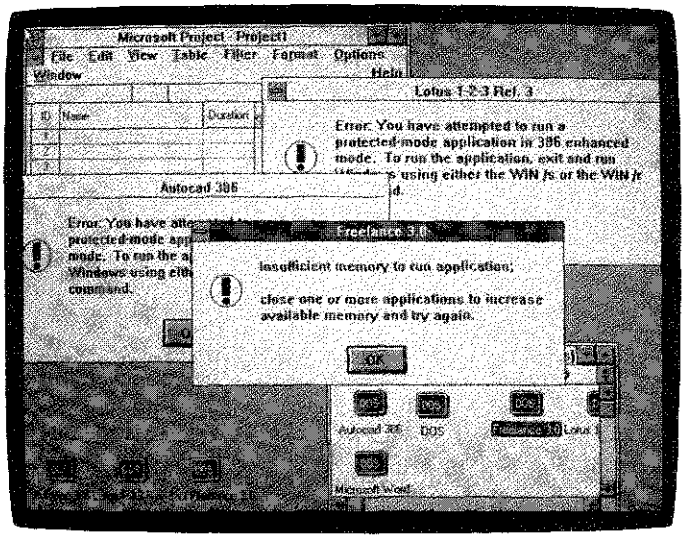
I've worked in two companies that use Unix. Some of my co-workers have been, and are, very smart Unix enthusiasts who know that DOS is crippled and useless. All of these folks have predicted the demise of DOS.

None of them has a personal computer at home. All of them drive a car that has a stick shift. A stick shift isn't the best human-automobile interface, but it's the one everybody learns and uses. OS/2 sales have been about equal to those of Chrysler's left-hand push-button cars, and Unix sales to 18-wheeler synchromesh transmissions. ■

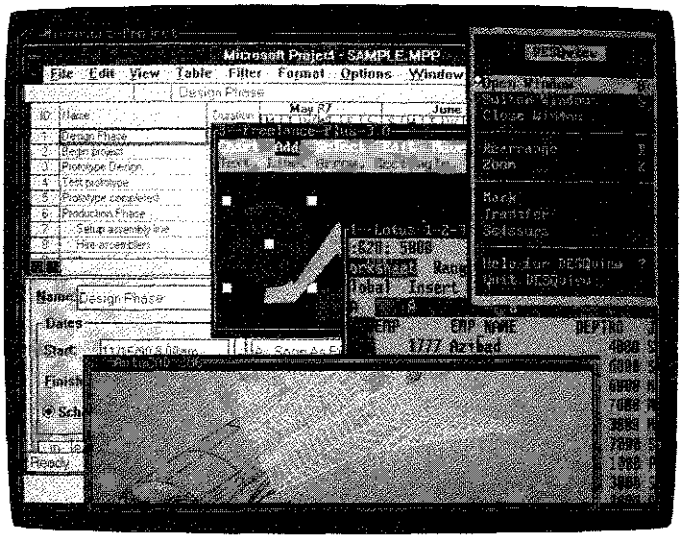
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Is Windows 3.0 the end of DESQview™?

Not if you're still using DOS programs.


Microsoft Project - Project1
File Edit View Table Filter Format Options Window Help
Lotus 1-2-3 Rel. 3
Error: You have attempted to run a protected-mode application in 386 enhanced mode. To run the application, exit and run Windows using either the WIN /s or the WIN /r command.
Autocad 386
Freelance 3.0
Insufficient memory to run application; close one or more applications to increase available memory and try again.
OK
Autocad 386
DOS
Freelance 3.0
Lotus
Microsoft Word

Windows 3.0. The multitasking, windowing environment.



DESQview 386 2.3. The multitasking, windowing environment.

Are you still using MS-DOS programs on your PC?

You may want to use DESQview as your primary operating environment.

The new DESQview 2.3 and DESQview 386 2.3 let you use your favorite DOS and DOS-extended programs in windows side-by-side on 80286, 80386 and i486 PCs. As you can see above, you can even run Windows programs within DESQview 386. So the next time you get error messages like the ones at the left in Windows, remember how the same set of programs look running in DESQview.

Whatever standard you use—DOS, extended DOS or Windows—DESQview is still the best way to get the most out of the hardware and software you own today.

DESQview. The obvious choice.

Quarterdeck

Quarterdeck Office Systems, 150 Pico Boulevard, Santa Monica, CA 90405 (213) 392-9851 Fax: (213) 399-3802

This comparison was made using a system like the one you might run: Both shots show an ALR FlexCache 33/386 running DOS 3.3 with VGA display adaptor, Novell NetWare v3.01 Rev. A, with IPX/SPX v3.01 Rev A, Microsoft Mouse 7.00, and Microsoft SMARTDrive v3.03 disk cache. Buffers were set to 20. For the Windows screen, we ran Microsoft Windows 3 HIMEM.SYS and EMM386.SYS. For the DESQview screen, we ran QEMM 386 v5.1.

Trademarks: Microsoft, Windows, MS-DOS, 80386, i486, ALR, FlexCache, Novell, NetWare. ©1990 Quarterdeck Office Systems

Circle 55 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

"Who would you expect to give you the best compiler for creating Windows applications?"

—Answer below

Introducing a vastly superior way to write applications for Microsoft® Windows. New Borland® C++. The only complete C and C++ programming environment for DOS *and* Windows. Borland C++ generates Windows applications and DLLs,* supports Windows debugging, and includes a visual resource editor. So you don't need to buy the Microsoft Windows Software Development Kit (SDK).

Borland C++ is from the people who know what professional programmers want. In fact, Turbo C++ Professional (the predecessor to Borland C++) won both *PC Magazine*'s 1990 Technical Excellence Award and *BYTE*'s 1990 Award of Excellence.

Designed for professionals by professionals

Every copy of Borland C++ comes with:

- ANSI C compiler and C++ compiler
- Turbo Drive™ compiler and integrated development environment running in protected mode for huge capacity
- Precompiled headers that significantly increase recompilation speed
- Turbo Debugger® for DOS and Windows
- Whitewater™ Resource Toolkit for visually creating icons, dialogs, bitmaps and menu bars
- Turbo Profiler™ and Turbo Assembler®

Compare these costs

	Microsoft® C 6.0	**Borland C++**
C compiler	$ 495	$495
C++ compiler	not available	included
Windows programming tools	$ 495 (SDK)	included
Assembler	$ 150	included
Total**	$1,140	**$495**

Buy now! Get *"Programming Windows"* free!

Buy Borland C++ now from your local dealer and get a free copy of *Programming Windows* by Charles Petzold, the best-selling book about learning Windows programming.†

Special upgrade pricing is available from Borland for owners of Turbo C,® Turbo C® Professional, Turbo C++ and Turbo C++ Professional.

See your dealer today, or call Borland to upgrade at 1-800-331-0877

Answer:

BORLAND

The Leader in Object-Oriented Programming

CODE MC51

*DLL = Dynamic Link Libraries. **$495 is the suggested retail price. †Offer good in U.S. and Canada while supplies last. Upgrade customers not eligible for free book. Copyright © 1991 Borland BI 1397

Circle **56** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED